13 Java EE .NET Security Interoperability

Security by Default

Security exploits and vulnerabilities are often causes of huge financial loss and disruption of business services. The Computer Security Institute (refer to [CSI] for details) has reported a worldwide financial loss of circa US$130 million that resulted from virus, unauthorized access, and theft of proprietary information in 2005, a US$7.3 million loss (compared to US$65 million loss in 2003) due to denial of service attacks, and an average US$355,552 (2005) loss per incident for proprietary information theft in 2003. A business application that was considered "secure" running on a Unix or Windows platform (for example, protected by firewall and anti-virus application) is not necessarily vulnerability-free when exchanging sensitive business data with another business application running on a different platform. This is because the interoperable solution is exposed to security vulnerabilities if one of the applications (either the sender or recipient) is exploited or is being attacked by hackers.

Fisher_CH13 3/24/06 5:05 PM Page 404

There are historic incidents of vulnerabilities in the Windows platform (such as flaw authentication [WindowsAuthFlaw]) or Java platform (such as a flaw in the JVM in [JavaVMFlaw]). These incidents are critical and can become the "Achilles' heel" (a critical problem that causes financial loss or disruption to the business service) for the mission-critical Java EE .NET interoperable solutions. Although the individual vulnerability incident may not be a direct root cause to security exploits of a Java EE .NET interoperable solution, any vulnerability exposed on either Solaris OE, Unix, Linux, or Windows platform becomes a "weakest link" to the security of the interoperable solution.

Web Services Interoperability (WS-I) identifies the following security threats that can impact Java EE .NET interoperability:

- **Message alteration** changing the message header or body during the transit.
- **Attachment alteration** changing the SOAP attachment during the transit.
- **Confidentiality** the capability to ensure no unauthorized access is made to the message.
- **Falsified messages** the message is falsified by using a different identity of the sender.
- **Man-in-the-middle** the message is being spoofed or tampered with during transit.
- **Principal spoofing** the information about the user or subject is being spoofed during transit.
- **Repudiation** the sender or recipient denied or repudiated about the message being sent or received.
- **Forged claims** the claim about sending the message is forged by tampering with the message content.
- **Message replay (or replay of message parts)** the message was once spoofed and modified for resending the message.
- **Denial of service** a malicious action to replay a message continuously or to overload the target service provider until the service provider is out of service.

To make a Java EE .NET interoperable solution *secure by default*, security architects and developers should consider the following security requirements. Also refer to [WSI-countermeasure] for the details of security scenarios and the counter-measures to the security threats.

Fisher_CH13 3/24/06 5:05 PM Page 405

- **Always Customize Security Settings** Do not take the default security settings of vendor products in the operating environment. Many business applications are not designed and deployed with security by default—they are designed with unused system services turned on when deployed, which may be open to security exploits and vulnerabilities that can severely impact the interoperable solution.
- **Use Open Standards for Interoperability** Web services security is currently an open standard for SOAP-based Web services. WS-I Basic Security Profile (BSP) 1.0 addresses these security threats. In essence, BSP 1.0 extends Web services security to handle SOAP attachments. These standards ensure that the applications are interoperable.
- **Use Strong Authentication Mechanisms**.
- **Use Secure Transport Mechanisms** Use of secure transport mechanisms such as SSL/TLS should address principal spoofing.
- **Use Digital Signature** Use of digital signature should address the security risks of message alteration, attachment alteration, confidentiality, repudiation, and forged claims. Signing the SOAP message header once, creation time, and optional user data over secure transport layer such as SSL/TLS are able to address the security risk of message replay.
- **Use Encryption** Use of encryption should address the security risks of confidentiality.

This chapter recapitulates the features of Java and .NET security that make interoperability easier. It also discusses different technologies (such as authentication in the Presentation tier) and the open standards (such as Web services security) where Java and .NET applications can interact. Finally, two interoperability strategies are discussed.

Java Security by Design

Application security is critical in Java technology. The Java runtime environment (aka JVM) provides a tightly guarded security environment for runtime execution. (Refer to [J2EE14], [J2EE14Tutor], and [LiGong] for more details.) Figure 13-1 depicts a high-level security overview. Inside the JRE, the Security Manager is responsible for code runtime verification and access control. The code runtime verification is managed by the Protection Domain, where different class files (namely, bootstrap class, system class, and user class) are verified by bootstrap class loader, system class loader, class loader,

Fisher_CH13 3/24/06 5:05 PM Page 406

and the bytecode verifier. The Access Module is responsible to authenticate and authorize the principal (user or service requester) against the security policy files (namely, java.security and java.policy files). The JRE supports a variety of authentication mechanisms, including JAAS login module, database security (using JDBC), or LDAP (using JNDI).

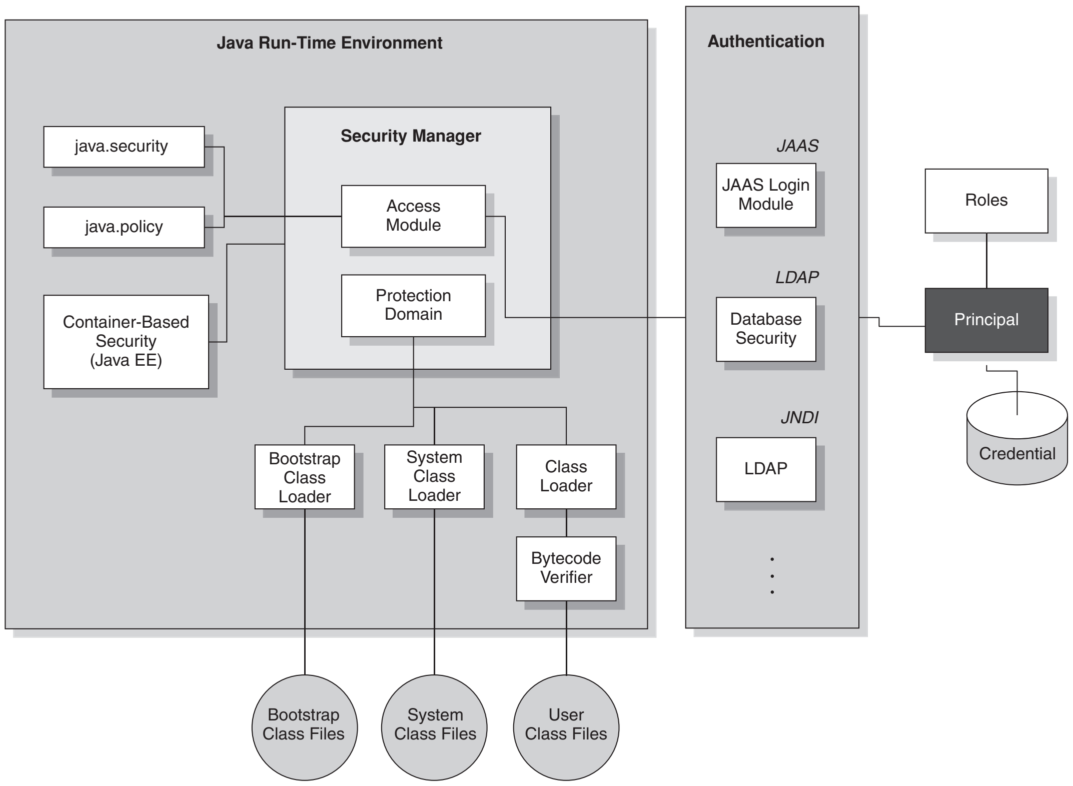


Figure 13-1
Java security overview

Java Runtime Security

Java SE provides a Java "sandbox" to restrict applets from accessing file systems and networks and untrusted applets from accessing all of the API functionality. The "sandbox" security architecture (refer to Figure 13-1) consists of three key components:

- **Bytecode Verifier** It verifies Java bytecodes that are compiled from Java source codes prior to execution. It ensures that the bytecodes do not violate permission policies or access system resources using incorrect type information.
- **Class Loader** The "primordial" (root) class loader bootstraps the class loading process and protects the runtime environment from loading harmful codes locally or remotely by hackers during the class loading process (so-called "code-spoofing"). It loads the initial classes required by all Java programs. The secure class loader `java.security.SecureClassLoader` then kicks off assembling and loading other classes locally, for example, bootstrap class files, system class files, and user class files.

  Class loading works under a class loader hierarchy. A child class must delegate to the parent class to load a specific class. If the parent class cannot load the specific class, then the child class loads it. Remote classes from the network are instantiated and loaded by the parent class loader as a new class. Thus hackers are not able to spoof attack by loading a malicious class directly into the JVM. For instance, hackers are able to insert a malicious version of `System.out.print` or `java.lang.String` into the application because the class loader loads a local version of `System.out.print` or `java.lang.String` under this class loading process. A Java archive (JAR) signer is a utility that seals packages for protection from tampering and verifies remote classes prior to loading to the JVM.
- **Security Manager** The security manager (`java.lang.Security Manager`) performs runtime checks on any method or any code accessing sensitive system resources (for example, file or network access), and generates a security exception for any security policy violation. It delegates the permission check to the `java.security.Access Controller` by calling the `checkPermission` method. The security manager can be invoked by specifying the system property while starting the JVM (for example, `java -Djava.security.manager myApps.class`) or creating an instance in the program code (for example, `System.setSecurityManager(new SecurityManager());`)

The security manager has a security policy database where security policies are maintained. The security policy database stores permission rules for authorization and key stores for authentication. A security policy relates to a set of permissions for a domain (system or application), which encloses a set of classes. Developers can also customize any additional protection of resources within the domain boundary, say, using the `SignedObject` class.

The **protection domain** (`java.security.ProtectionDomain` class) is another important security concept. It refers to the system and application components (for example, a group of classes) of the runtime environment that can be secured according to the predefined security policy. There are two types of protection domain: **static** (grant permissions specified only when constructed) and **dynamic** (grant permissions specified when constructed and permissions granted by the security policy). The protection domain extends the "sandbox" security architecture by associating a group of principals with permissions. For example, a protection domain associates permissions with a **code source** (URL where the class file comes from), such that any classes originating from the same URL will have the same signature and key placed in the same domain, and thus granted the same permissions and access rights. This enhances the current mechanism to load a class. Moreover, the security policy allows defining an association between the principals and permissions for the classes (via the code source). It can be passed as a parameter to the application, which can use different protection domains wherever necessary.

Java SE security introduces a set of additional security concepts (see [LiGong] for details), which includes the concepts of a **Principal** (an entity that a security service can authenticate with an authentication protocol) and **security domain** (the scope related to a set of security policies defined by the administrator of a security service). There are two important files under JVM that store security policy information: `java.policy` and `java.security` under the directory, %J2SE%/lib/security, where %J2SE% is the file location for the JVM.

Authentication Mechanisms

Figure 13-1 shows an example of three different authentication mechanisms. A subject (user) has multiple principals, or multiple user names or identities. Suppose one of his principals (using the `java.security.Principal` class) possesses a digital certificate as a credential. Under the Java SE security architecture, the principal can use the credential to authenticate with the applications via JAAS (using the `javax.security.auth.login.*` class), JNDI (using the LDAP directory server), or JDBC (using back-end database security). In this example, the security manager is used. Upon successful authentication, the security manager will check permission, and pass control to the access module. The access module (using the `java.security.AccessController` class) checks permission by checking the `java.security` file, which contains the policy URL (`policy.url.1`) and

Fisher_CH13 3/24/06 5:05 PM Page 409

keystore information (`keystore.type`). The `java.security` file associates the permission (`grant ... permission ...`) with the principals. If the principal-permission relationship is found, then the access module grants access to the principal for the application resources to the principal.

Apart from the "sandbox" security architecture, Java SE also provides some authentication and encryption services that work with JCA and JCE layers. These security services include JAAS, JSSE, JGSS, and CertPath. Applications or security packages can also customize their security APIs using these security services.

Container-Based Security

The Java EE container provides a comprehensive application-level security that is related to the application component deployment and runtime environment.

- **Declarative Security** According to [J2EE14], Java EE security has the notion of declarative security. In other words, the application's security structure, including security roles, access control, and authentication requirements in a form external to the application, is expressed in the deployment descriptor. The **deployment descriptor** is in effect a contract between the application component provider and the deployer or application assembler, where the application security policy is mapped to the security structure of the relevant operating environment.
- **Programmatic Security** Java EE security architecture also provides some APIs (programmatic security) to manipulate the roles and principals, in addition to the declarative security. This supports both servlets (`isUserInRole` and `getUserPrincipal` using the interface `HttpServletRequest`) and EJBs (`isCallerInRole` and `getCallerPrincipal` using the interface `EJBContext`).

Security Interoperability Features

Java technology has provided several interoperability features to secure business applications. The following highlights a few major security interoperability features:

- "Building block" security components to support interoperability, for example, JAAS and JSSE.

- JSR implementation that enables interoperability, for example, JAX-RPC 2.0 and WS-I Basic Security Profile 1.0/1.1.
- Support of single sign-on using Web SSO protocol, WS-MEX protocol, Liberty, and SAML.
- Support of security interoperability standards, for example, OASIS's Web services security.
- Support of WS-Policy by Java Web Services Developer Pack 2.1 or later.

.NET Security by Design

.NET security is targeted at developers. The .NET Framework provides a developer-centric and runtime security model on top of the Windows operating system security. It supports a *role-based security* that defines the access rights for resources using a role or a group. Role-based security addresses the security risk of broken access control for applications. At the software code level, the .NET Framework has **Code Access Security**, also known as **evidence-based security**, that defines whether or not a user can be trusted to access a resource. Code Access Security addresses the security risk of tampering or the use of a Trojan horse when downloading codes from external sources or Internet. Refer to [Watkins] for an overview of the .NET Framework security.

Figure 13-2 depicts a high-level security overview of the .NET Framework. The .NET Common Language Runtime (CLR) provides a runtime environment under the Windows hosting environment. When a .NET application is deployed, the .NET Framework assembles and deploys the .NET application to the target runtime environment in MSIL (Microsoft Intermediate Language) with the associated metadata. MSIL is an object-oriented assembly language that can be compiled to x86 native codes by a just-in-time compiler for execution in the CLR environment. **metadata** is a set of tables, also known as contract or blueprint, that depict the assembly's types, their methods, fields, signatures, and dependencies on other assemblies.

The .NET Common Language Runtime also provides a runtime security system that uses a policy manager to evaluate what permission should be granted to a service request. A Principal interacts with a .NET application and issues a service request to access resources. The security system in the CLR evaluates the service request based on the evidence, which is a set of information that constitutes input to security policy decisions, for example,

Fisher_CH13 3/24/06 5:05 PM Page 411

origin of the codes and digital signature of the assembly, in the Windows hosting environment and the security policies defined in the CLR's policy levels and permission set.

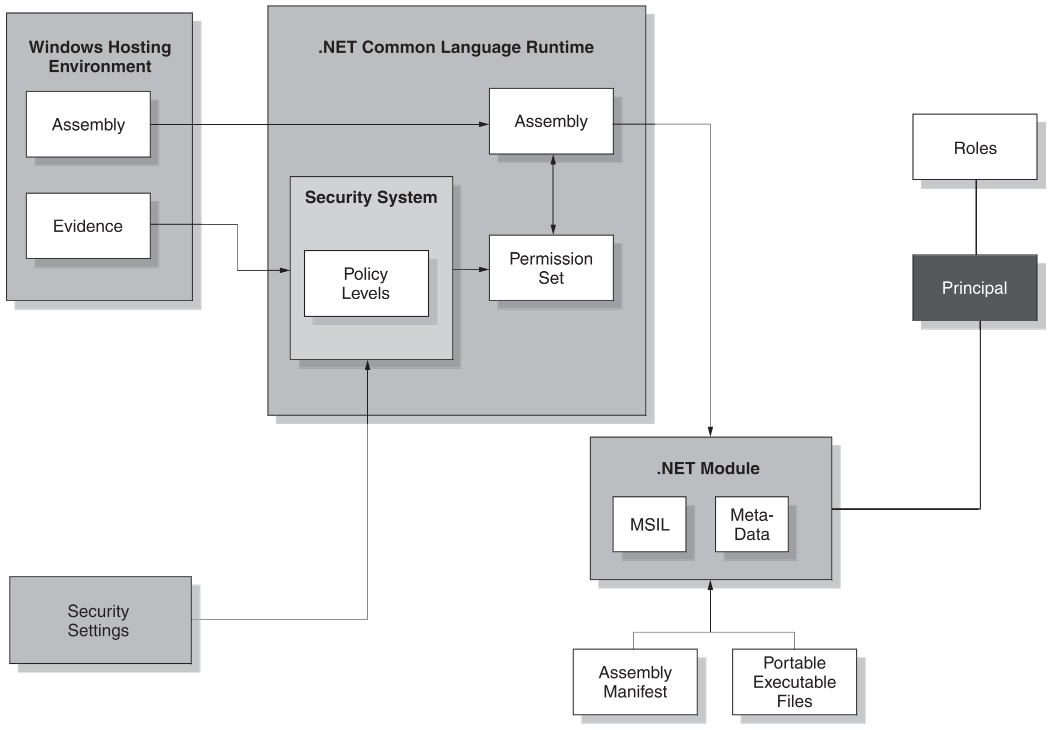


Figure 13-2
.NET security overview

A .NET application is composed of an assembly and one or more .NET modules. An **assembly** is the unit of code deployment in the .NET Common Language Runtime environment. It consists of an **assembly manifest** (a list of the assembly layout and global attributes) and one or more .NET modules. .NET modules are either DLLs or EXE Windows portable executable files. They contain the Microsoft Intermediate Language (MSIL), the associated metadata, and optionally the assembly manifest.

In the .NET CLR environment, an assembly uses the basic permission set class `System.Security.PermissionSet` to grant permissions to codes that are defined in the policy levels of the security system. Permissions can be code access permissions, which protect the resources directly, or identity permissions, which represent evidence that is granted to assemblies.

Fisher_CH13 3/24/06 5:05 PM Page 412

A Principal authenticates with the Windows system and invokes a .NET application to access a system resource. The CLR environment evaluates the security policies to determine whether the Principal has appropriate permission and access rights to execute the program codes and access the system resources.

Code Access Security

Code Access Security (CAS) is a key security feature of .NET Framework. It supports the requirement that different code should have different levels of trust. Using CAS, the security access control is based on the identity of the code, not individual user identity (such as user id), who executes or runs the software codes. This addresses the limitation of access control for different software codes by username-password, which is at a coarse granularity level. For example, developers can define code access security policies to constrain the ability of an assembly to perform file input/output and restrict file input/output to a specific directory. Code access security addresses the insufficiency of guarding against malicious or faulty codes that may have been downloaded from e-mail or the Internet that can damage files, though the user has already been authenticated and authorized to run the executable codes. In other words, CAS addresses the gap of protection against malicious codes and is complementary to role-based security.

CAS has three key elements: evidence, security policy, and permissions. Evidence refers to the set of information that constitutes input to security policy decision. This includes the characteristics of an assembly, such as the Web site from where an assembly is loaded. Security policy (also refer to next section for details) is a set of rules used by the runtime **policy resolution process**, also known as the Policy Decision Point, to determine which permissions an assembly can be granted. Permissions refers to the authority of an assembly's code to access protected operations and resources. There are three different permission classes in .NET: code-access permissions, for example, file input/output access granted to an assembly; identity permissions, where an assembly presents a certain host evidence value to the runtime policy resolution process as "identity;" and role-based permissions, when access is granted to a role, such as system administrator.

To illustrate how CAS works, consider a sample .NET application, "myinterop.exe," by re-using the architecture diagram in Figure 13-2. When a user runs the application "myinterop.exe," the .NET Common Language Runtime loads the "myinterop.exe" assembly from the Windows hosting environment. The runtime then evaluates its evidence and determines what

Fisher_CH13 3/24/06 5:05 PM Page 413

permissions to grant the application against the defined security policies. For instance, the application, myinterop.exe, has a permission request to write to the text file "userpassword" in the local hard drive. The runtime's policy resolution process determines what permission should be granted to the assembly based on the security policies as well as the permission set assigned to the assembly (for example, `FileIOPermission` object defined in the myinterop.exe assembly). Once the runtime confirms that the application has the necessary permission to write to the text file, "userpassword," the runtime responds with a positive result to the `File.Write` method. Otherwise, the runtime throws a `System.Security.SecurityException` if the permission is not granted.

CAS is about the understanding of the relationship between evidence, policies, and permission, the details of which are beyond the scope of this chapter. Please refer to the References section for more resources.

Security Policies

Security policies in the .NET Framework refer to the mechanism for administrators to express the level of trust for different codes. There are four key elements of the .NET security policies:

- **Membership conditions** A membership condition resembles an object that answers "yes" or "no" when asked if an assembly matches its membership test. The membership conditions turn the evidence of an assembly into a grant set
- **Code groups** Code groups map the .NET Framework code to specific levels of trust. They are bindings between membership conditions and permission sets. If code matches the membership condition in a code group, it is ranked a permission set.
- **Policy levels** The `System.Security.Policy.PolicyLevel` class defines policy levels using a list of named permission sets, a code group hierarchy, and a list of "full trust" assemblies. There are four policy levels supported: enterprise, machine, user, and application domain. During the policy resolution of an assembly, the Policy Manager evaluates the assembly's evidence against each individual policy level via the `SecurityManager.ResolvePolicy` method.
- **Default security policy** This is the culmination of the default policies of all four policy levels, where each policy level has a hard-coded default. All default policy levels are identical with reference to the permission set lists and assembly lists. The permission set lists contain all the named permission sets.

Fisher_CH13 3/24/06 5:05 PM Page 414

Execution-Time Security

When an assembly is deployed to a target machine, the Assembly Loader loads the assembly in the CLR environment with the context of a trusted host, that is, the host is the trusted piece of code that is responsible to launch the runtime. The Policy Manager evaluates the current security policy, the evidence known about the assembly, and the set of permission requests, if any, made in the assembly metadata. It determines what permissions should be granted to the service requester based on the security policies for code access. Upon evaluation by the Policy Manager, the Class Loader loads the class for the JIT compiler to verify the codes prior to execution. The Code Manager then translates the classes into native code for execution.

Security Interoperability Features

.NET technology has provided several interoperability features to secure business applications. The following highlights a few major security interoperability features:

- Support of WS-I Basic Security Profile 1.0 via Web Services Enhancement (WSE).
- Support of single sign-on using Web SSO protocol via Active Directory Federation Services and the WS-MEX (metadata Exchange) protocol via Microsoft Windows Communication Foundation (WCF) or formerly Indigo.
- Support of security interoperability standards, for example, OASIS's Web Services Security.
- Support of WS-Policy. WSE is an add-on to the .NET Framework and provides a policy editor that allows defining policies for Web services using WS-Policy.

Security for Interoperability

Previous chapters have discussed that Java and .NET applications can interoperate synchronously or asynchronously in different architecture tiers. As security is end-to-end, security for interoperability should not be limited to a single application component (for example, .NET bridge) or a specific architecture tier (for example, Web tier). Further, the security requirements for interoperability in each architecture tier are different. This section discusses

Fisher_CH13 3/24/06 5:05 PM Page 415

the security requirements and what enabling technologies and security standards are available to address these requirements. The details of security standards for interoperability are discussed in the next section. Adopting interoperability technologies that support security standards allows wider choice of vendor products and easier implementation.

Figure 13-3 depicts the areas of security for interoperability that have support for security standards such as WS-I Basic Security Profile and WS-Security. A Java client should be able to perform a single sign-on with the .NET application and similarly for a .NET client with a Java application. To ensure client-to-server communication is secure, developers can use HTTPS or SSL/TLS to encrypt the communication channel.

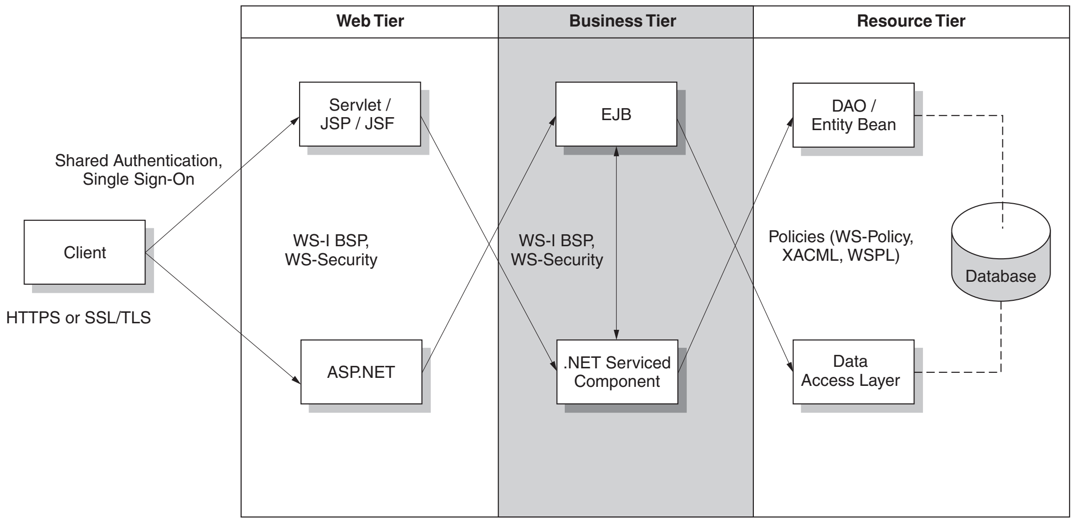


Figure 13-3
Security for interoperability

In both the Web and the Business tiers, the client should be able to initiate service requests or exchange business data synchronously or asynchronously using Web services (with WS-I BSP and WS-Security). This should allow a servlet, JSP, or JSF component under the Web tier to interoperate with a .NET service component under the Business tier—or an ASP.NET page under the Web tier to interoperate with an EJB object under the Business tier.

In the Resource tier, a Java servlet or EJB component can also request access to resources such as business data and database objects implemented by means of the Data Access Layer using a policy language such as

Fisher_CH13 3/24/06 5:05 PM Page 416

WS-Policy, XACML, and Web Services Policy Language (refer to next section for more details). Similarly, a .NET-serviced component can also share the same policy language (provided that they are available in both .NET and Java language) when requesting access to resources via Data Access Objects or Entity Beans.

There are always business scenarios in the existing or legacy environment that Figure 13-3 does not cover. As most of these scenarios use nonstandard interoperability technologies that are usually proprietary or highly customized on a case-by-case basis, they require additional cost and efforts to analyze the potential security risks and to mitigate the vulnerabilities. For example, if a bridge technology is used for Java EE .NET interoperability, developers and security architects need to analyze the risks of the bridge technology and the customized application codes that connect to the bridge. The bridge would become an easier target for hacking or the single point of failure attacks. Even though the security for the bridge is strong, there may be considerable unknown risks for the customized application codes related to it.

The following sections elaborate on the security requirements of the Java EE .NET interoperability and discuss the technologies available to mitigate the security risks.

Secure Transport

The client-to-server session is often a target of security spoofing. A basic security requirement for interoperability when exchanging user credentials or sensitive business data is secure data transport. Java and .NET applications can use HTTPS or SSL/TLS in the data transport layer to secure the client-to-server session. Secure data transport can ensure confidentiality and reduce the risk of principal spoofing.

Security Interoperability by Tiers

Interoperability at the Web Tier

A Java client can authenticate with an Active Directory or a Directory Server using a JAAS login module. Similarly, a .NET client can authenticate with a Directory Server. Digital certificates can be used as a common user credential; however, Java and .NET clients need to create shared session state in order to interoperate in a heterogeneous environment. The capability to

Fisher_CH13 3/24/06 5:05 PM Page 417

authenticate with both Java and .NET application servers and to create shared common session data are key security requirements for interoperability. These security requirements allow the Principal to share session information between the Java and .NET environments and not necessitate relogin.

A Java client can authenticate with a .NET application using the shared authentication approach. Similarly, a .NET client can also authenticate with a Java application using the same shared authentication approach. Shared authentication here refers to the use of form-based authentication and customization of shared session data for both Java and .NET applications. Form-based authentication allows page-level authentication to a Web application. Shared session data can be stored in a customized shared session state database or a Directory Server using existing session APIs in both Java and .NET platforms—for example, Java has many APIs under the `javax.servlet.http.HttpSession` class, and .NET has `Shared Session` object.

Nevertheless, customized processing logic for shared authentication and shared session data are often proprietary and are specific to certain implementation. The use of Web SSO MEX (Single Sign-on metadata Exchange) protocols is a proposed standard for Java EE .NET interoperability to achieve single sign-on and should be recommended. Please refer to the following section for more details.

Shared authentication, shared session data, and single sign-on using Web SSO MEX protocol are mechanisms to address broken authentication and session management. They also rely on strong authentication mechanisms, for example, use of digital certificates and strong user passwords, and reliable authentication infrastructure such as Directory Server. For Web services such as asynchronous SOAP messages, it is critical to use the WS-I Basic Security Profile (BSP) and WS-Security standards. WS-I BSP ensures that both Java and .NET applications are using a common semantic for SOAP messages. WS-Security supports service requests or replies that are digitally signed and/or encrypted to ensure confidentiality. It addresses the risk of message alteration, attachment alteration, falsified messages, repudiation, forged claims, and message replay.

Interoperability at the Business Tier

Security interoperability requirements for the Business tier are similar to those for the Web tier. The key difference is that the Business tier interaction is often server-to-server, not client-to-server. Interaction between .NET-serviced components and EJBs or service requests from the Web tier are often

Fisher_CH13 3/24/06 5:05 PM Page 418

point-to-point, and these components do not use SSL/TLS, which is used for client-to-server communication. Customized and point-to-point security processing logic (for example, encrypted transactions) that secures the business transactions in the Business tier is usually proprietary and is unlikely to be reusable in another environment. For interoperability using Web services, WS-I BSP and WS-Security provide an open standard to secure business transactions. For interoperability using a bridge technology, developers need to rely on customized encryption or proprietary security mechanism.

Interoperability at the Resource Tier

Both Java and .NET platforms support a variety of access control mechanisms to determine which resources are accessible to the service requesters. They may either embed the access control processing logic into the programming codes or use a policy framework. Using a policy framework, developers can decouple the custom access control processing logic from the application codes. Because the access control processing logic is separate from the application codes, it will be more flexible, allowing management of changes without the need to recompile or retest the entire application for every security policy change.

Without a common policy framework, an interoperability solution needs to rely on two different access control systems. If the two policy frameworks are "out of sync," the service requester may be denied access. It is also time-consuming to troubleshoot which side of the policy framework is problematic.

The use of common and standard security policies, such as WS-Policy and XACML, across Java and .NET platforms addresses the risk of broken access control. For example, Microsoft's Web Services Enhancement provides support for WS-Policy, and Sun's XACML Kit is available on both Java and .NET platforms. Please refer to the next section for details on these policy frameworks.

Support of Audit Control and Compliance

In recent years, support of audit control and compliance (for example, Sarbanes-Oxley, Gramm-Leach-Bliley Act of 1994, HIPPA or Health Insurance Privacy and Portability Act of 1996) has become a key security interoperability requirement. These compliance requirements are focused in building the capability for tracking unusual or suspicious user activities, ranging from unauthorized access to suspicious high volume business transactions. They

Fisher_CH13 3/24/06 5:05 PM Page 419

also require a timely report of the unusual or suspicious security events, and tracing the source of the service requesters. Thus, it is extremely important that the Java and .NET interoperability design should be able to have risk mitigation mechanisms for security attacks, and to build up capability for "track and trace" of any unusual security events and service requests.

Security Standards for Interoperability

One of the purposes of creating security standards is the ease of interoperability. Using security standards, developers have more flexibility to interoperate between Java and .NET applications and do not lock in with a specific vendor product. This section introduces security standards and specifications for interoperability using Web services with details: WS-Security, WS-I Basic Security Profile, XACML, WS-Policy, Web Services Policy Language, and Single Sign-on Metadata Exchange (SSO MEX). Table 13-1 provides a list of security standards and specifications that are relevant for Java EE .NET interoperability. Although WS-Policy and SSO MEX are not a security standards yet, they are important security specifications for Java EE .NET interoperability.

Table 13-1
List of Security Standards and Specifications for Interoperability

	Security Standards	Security Specifications
Application Security	Web Services Security (OASIS)	
	WSI Basic Security Profile (WSI)	
Policy	XACML (OASIS)	
	Web Services Policy Language (OASIS)	WS-Policy
Single Sign-on		SSO MEX
Others (not covered in this chapter)		WS-Trust
		WS-SecureConversation

Web Services Security

Web services security (WS-Security) is an approved standard to secure SOAP messages under OASIS. It leverages XML Digital Signature and XML Encryption for message integrity and confidentiality. WS-Security provides an abstract message security model that specifies how to protect a SOAP message in terms of a security token and digital signature. Security tokens are simply assertions of claims about the user identity and can be used to assert the binding between the authentication secrets, keys, or security identities. WS-Security currently supports a variety of security tokens, for example, user name token (aka password), binary token (such as the X.509v3 certificate or Kerberos ticket), and XML tokens.

From a Java EE .NET interoperability perspective, WS-Security plays a key role because it defines a standard format where a Java and .NET application can exchange business data. Listing 13-1 shows a .NET program excerpt how to sign a message using a X.509 certificate.

Listing 13-1
Sample .NET Program Excerpt Showing How to Sign a Message with an X.509 Certificate

```
using System;
using Microsoft.Web.Services2.Security;
using Microsoft.Web.Services2.Security.Tokens;
using Microsoft.Web.Services2.Security.X509;
using Microsoft.Web.Services2.QuickStart;

namespace X509SigningClient {
    /// <summary>
    /// This is a sample which allows the user to send a message to
    /// a Web
    /// service that has been signed with an x.509 certificate.
    /// </summary>
    class X509SigningClient : AppBase {

        [MTAThread]
        static void Main(string[] args) {

            X509SigningClient client = null;
            try {
                client = new X509SigningClient();
                client.Run();
            } catch (Exception ex) {
                Error(ex);
            }
            Console.WriteLine( "Sample - .NET X.509 Signing Client" );
            Console.WriteLine();
        }

        public void Run() {
            // Create and configure Web service proxy
            //
            //ManufactureDelegate serviceProxy = new
                 ManufactureDelegate(...);
            //ConfigureProxy(serviceProxy);
            ...
            // Fetch X.509 security token and generate asymmetric key
            // 'false' forces to use WSE sample certificate
            X509SecurityToken token =
                AppBase.GetClientToken(false);

            if (token == null)
                throw new ApplicationException
                    ("Cannot retrieve security token!");

            // Add the signature element to a security section on the
            // request
            // to sign the request
            serviceProxy.RequestSoapContext.Security.Tokens.Add(token);
            serviceProxy.RequestSoapContext.Security.Elements.
               Add(new MessageSignature(token));

            // Invoke business service
            ...
        }
    }
}
```

A Java platform provides low-level and fine-grained programming APIs that generate XML signatures using the JSR 105 (XML signature) API (refer to Listing 13-2 for an example). In practice, developers do not use these fine-grained APIs directly to sign a message in WS-Security. There are tools available to implement WS-Security with coarse-grained programming APIs. For example, JWSDP version 1.5 provides a useful WS-Security security handler, `SecurityEnvironmentHandler`, that refactors the encryption, decryption, and the digital signing processing logic into handlers. It does not require developers to embed the common security processing logic into the application. Developers can simply turn on or off the security processing logic in a configuration file (refer to the next section, "Secure Object Handler Strategy").

Listing 13-2
Sample Java Program Excerpt Generating a Digital Signature

```
import javax.xml.crypto.*;
import javax.xml.crypto.dsig.*;
import javax.xml.crypto.dom.*;
import javax.xml.crypto.dsig.dom.DOMSignContext;
import javax.xml.crypto.dsig.keyinfo.*;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.security.*;
import java.util.Arrays;
import java.util.Collections;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class SigningClient {

   public static void main(String[] args) throws Exception {

       // Create digital signature factory
       String providerName = System.getProperty
           ("jsr105Provider",
```

Fisher_CH13 3/24/06 5:05 PM Page 423

```
     "org.jcp.xml.dsig.internal.dom.XMLDSigRI");
XMLSignatureFactory dsigFactory =
    XMLSignatureFactory.getInstance("DOM",
    (Provider) Class.forName(providerName).newInstance());

// Create document factory and object reference using SHA1
// digest
Reference ref = dsigFactory.newReference("#object",
    dsigFactory.newDigestMethod(DigestMethod.SHA1, null));

DocumentBuilderFactory docFactory =
   DocumentBuilderFactory.newInstance();
docFactory.setNamespaceAware(true);
Document doc = docFactory.newDocumentBuilder().newDocument();
Node text = doc.createTextNode("PO number");

XMLStructure content = new DOMStructure(text);
XMLObject obj = dsigFactory.newXMLObject
    (Collections.singletonList(content), "object", null, null);

// Create the SignedInfo
SignedInfo signedInfo = dsigFactory.newSignedInfo(
    dsignFactory.newCanonicalizationMethod
    (CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS, null),
       dsignFactory.newSignatureMethod(SignatureMethod.DSA_SHA1,
       null),
    Collections.singletonList(ref));

// Create a DSA KeyPair
KeyPairGenerator keyPairGen =
   KeyPairGenerator.getInstance("DSA");
keyPairGen.initialize(512);
KeyPair keyPair = keyPairGen.generateKeyPair();

// create key value using DSA public key
KeyInfoFactory keyInfoFactory =
   dsigFactory.getKeyInfoFactory();
KeyValue keyValue =
   keyInfoFactory.newKeyValue(keyPair.getPublic());
KeyInfo keyInfo =
```

continues

Listing 13-2 (continued)

```
keyInfoFactory.newKeyInfo(Collections.singletonList(keyValue);

        // create XML signature
        XMLSignature signature = dsigFactory.newXMLSignature(signedInfo,
           keyInfo,Collections.singletonList(obj), null, null);

        // create context using DSA private key
        DOMSignContext signContext = new
           DOMSignContext(keyPair.getPrivate(), doc);

        // Generate enveloping signature using private key
        signature.sign(signContext);
        ...

    }
}
```

WS-I Basic Security Profile

Web Services Interoperability (WS-I)'s Basic Security Profile (BSP) version 1.0 (refer to [BSP] for details) is a draft specification that defines the semantics of using the elements of OASIS's Web Services Security and places constraints on its use to achieve interoperability. BSP includes the following key characteristics:

- **Elements of a Secure SOAP Message** BSP uses elements defined in the Web Services Security 1.0 specification and includes a secure envelope, secure message, security header, reference, digital signature, encrypted key, encryption reference list, encrypted key reference list, encrypted data, security token reference, internal security token, and timestamp.
- **Secure Transport Layer** HTTP over TLS 1.0/SSL 3.0 should be used for transport layer security. Though BSP does not prohibit use of any specific TLS or SSL ciphersuites, it recommends ciphersuites that support the AES encryption algorithm, for example, TLS_RSA_WITH_AES_128_CBC_SHA, and discourages ciphersuites that are vulnerable to man-in-the-middle attacks, for example, SSL_RSA_WITH_NULL_SHA.

Fisher_CH13 3/24/06 5:05 PM Page 425

- **SOAP Message Security** BSP places some constraints in the use of binary security tokens, for example, only `Base64Binary` encoding type is supported, and `<wsse:BinarySecurityToken>` with a single X.509 certificate in the element `<wsu:Id>` must have the `ValueType` value http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3. The `ValueType` element is interesting and important from the perspective of ensuring openness and interoperability because it is used to define customer bearer tokens, which is used, for instance, by the Liberty Alliance for single sign-on. Moreover, BSP defines the semantics of a creation timestamp (`<wsu:Timestamp>`) element to be used for each `<wsu:Created>` element but does not allow leap seconds. BSP also specifies the order of processing the signature and encryption blocks (that is, signature, encrypted key, and encryption reference list) within the `<wsse:Security>` headers so that the recipient gets the correct result by processing the elements in the order they appear.
- **Username Token Profile** BSP specifies the semantics regarding the use of a username token, for example, the `<Type>` attribute (such as `Type='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText'`) must be specified to avoid any ambiguity of the element, `<wsse:Password>`. Another example is that Web Services Security does not fully describe the proper `ValueType` for the username token, and BSP uses the value `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-messagesecurity-1.0#UsernameToken for the attribute wsse:SecurityTokenReference/wsse:Reference/@ValueType`.
- **X.509 Certificate Token Profile** Web Services Security supports three token types (namely, X509v3, x509PKIPathv1, and PKCS7) in the X.509 certificate token profile. BSP places some constraints to the profile, for example, when certificate path information is provided via either X509PKIPathv1 or PKCS7 formats, the sender must provide one of the X509PKIPathv1 or PKCS7 token types. In addition, when the element, `<wsse:KeyIdentifier>`, is used within a security token reference to denote an X.509 certificate token, the element, `<wsse:KeyIdentifier>`, must use the value `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier` in the `<ValueType>` attribute.

Fisher_CH13 3/24/06 5:05 PM Page 426

- **Use of XML Signature** BSP places some constraints on the use of XML Signature—for example, the enveloping signature is not allowed, and a detached signature should be used instead. BSP also recommends two key signature algorithms (hmac-sha1 and rsa-sha1) and one digest algorithm (SHA1) for interoperability.
- **Use of XML Encryption** BSP adds some constraints on the use of XML Encryption—for example, an encrypted key `<xenc:EncryptedKey>` must precede any encrypted data in the same security header.
- **Attachment Security** BSP specifies some constraints on the SOAP with Attachments for interoperability, one of which is that secure message must conform to WS-I Attachment Profile 1.0. BSP also defines the semantics around the signed attachments—for example, reference to a signed MIME part must use a URI attribute of the form `"cid:partToBeSigned"`, and encrypted attachments—for example, encrypted data referencing a MIME part must include a type attribute with the value of either `"...#Attachment-Content-Only"` or `"...#Attachment-Complete"`.

XACML

Business applications usually have custom security policies to determine which resources or business data a service requester can access. Some applications may tightly couple the access control processing logic with the business processing logic. This would make the access control policies very difficult to extend or customize. For Java and .NET applications running on different platforms, it is fairly important to have a generic, flexible, and adaptive policy framework that operates on both Java and .NET applications. When there is a change of access control policy, developers do not need to modify their program logic or recompile their programs.

eXtensible Access Control Markup Language (XACML) version 2.0 is an approved security policy management standard under OASIS (refer to [XACML2]). Currently, it has both Java and .NET implementations. XACML is both a policy language and an access control decision request-response language encoded in XML. It is used to express authorization rules and polices and to evaluate rules and policies for authorization decisions. Moreover, XACML is used to make authorization decision requests and responses.

In a typical application environment, a user wants to make a request to access certain resources. The Policy Enforcement Point (PEP) is a system or

Fisher_CH13 3/24/06 5:05 PM Page 427

application that protects the resources. The PEP needs to check whether the service requester is eligible to access the resources. It sends the resources request to the Policy Decision Point (PDP), which looks up the security access control policies. XACML provides both a policy language and an access control decision request-response language to meet the security access control requirements. With XACML, the PEP forms a query language to ask the PDP whether or not a given action should be allowed. The PDP responds by returning the value of either `Permit`, `Deny`, `Indeterminate` (decision cannot be made due to some errors or missing values), or `Not Applicable` (the request cannot be answered by this service).

XACML Components

XACML provides a rich policy language data model to define flexible and sophisticated security policies. The following summarizes the key components in an XACML data model, [XACML2]:

- **Policies** A policy represents a single access control policy, expressed through a set of rules. Policies are a set of rules together with a rule-combining algorithm and an optional set of obligations. Obligations are operations specified in a policy or policy set that should be performed in conjunction with enforcing an authorization decision. Each XACML policy document contains exactly one Policy or PolicySet root XML tag.
- **Policy Set** A Policy Set is a set of policies or other Policy Sets and a policy-combining algorithm, and a set of optional obligations.
- **Rules** Rules are expressions describing conditions under which resource access requests are to be allowed or denied. They apply to the target (`<Target>`), which can specify some combination of particular resources, subjects, or actions. Each rule has an effect (which can be `permit` or `deny`) that is the result to be returned if the rule's target and conditions are true. Rules can specify a condition (`<Condition>`) using Boolean expressions and a large set of comparison and data-manipulation functions over subject, resource, action, and environment attributes.
- **Target** A Target is basically a set of simplified conditions for the Subject, Resource, and Action that must be met for a PolicySet, Policy, or Rule to apply to a given request. These use Boolean functions (explained more in the next section) to compare values found in a request with those included in the Target. If all the conditions of a Target are met, then its associated PolicySet, Policy, or Rule applies to the request. In addition to being a way to check applicability, Target

information also provides a way to index policies, which is useful if several policies need to be stored and then quickly sifted through to find which ones apply.

- **Attributes** Attributes are named values of known types that may include an issue identifier or an issue date and time. Specifically, attributes are characteristics of the Subject, Resource, Action, or Environment in which the access request is made. For example, a user's name, their group membership, a file they want to access, and the time of day are all attribute values. When a request is sent from a PEP to a PDP, that request is formed almost exclusively of attributes, which are compared to attribute values in a policy to make the access decisions.

Example

Listing 13-3 depicts a service request to access the URL http://www.supply-chain.com/purchaseorder.html. The service requester is a buyer with the subject buyer@javadotnetinterop.com and the access rights group `tradingPartner`.

Listing 13-3
Sample Service Request to Access a URL

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Subject>
      <Attribute
          AttributeId=
   "urn:oasis:names:tc:xacml:2.0:subject:subject-id"
          DataType=
"urn:oasis:names:tc:xacml:2.0:data-type:rfc822Name">
        <AttributeValue>
            buyer@javadotnetinterop.com
        </AttributeValue>
      </Attribute>
      <Attribute
          AttributeId="group"
               DataType=
            http://www.w3.org/2001/XMLSchema#string
               Issuer=
```

Fisher_CH13 3/24/06 5:05 PM Page 429

```
               "admin@javadotnetinterop.com">
        <AttributeValue>tradingPartner</AttributeValue>
      </Attribute>
    </Subject>
    <Resource>
      <Attribute AttributeId=
      "urn:oasis:names:tc:xacml:2.0:resource:resource-id"
                DataType=
           "http://www.w3.org/2001/XMLSchema#anyURI">
        <AttributeValue>
http://www.supplychain.com/purchaseorder.html
</AttributeValue>
      </Attribute>
    </Resource>
    <Action>
      <Attribute
          AttributeId=
     "urn:oasis:names:tc:xacml:2.0:action:action-id"
          DataType=
     "http://www.w3.org/2001/XMLSchema#string">
        <AttributeValue>execute</AttributeValue>
      </Attribute>
    </Action>
  </Request>
```

Listing 13-4 defines the security policy in XACML. The policy indicates that only service requesters with the e-mail address `javadotnetinterop.com` and the access rights group `tradingPartner` can access the URL, http://www.supplychain.com/purchaseorder.html.

Listing 13-4
Sample XACML Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="MemberCanPlaceOrder_ObligationPolicy"
        RuleCombiningAlgId=
        "urn:oasis:names:tc:xacml:2.0:rule-combining-algorithm:permit-
overrides">
```

continues

Listing 13-4 (continued)

```
  <Description>
     This policy states that trading partners with a valid domain name
     @javadotnetinterop.com should be able to submit purchase order
     online using the URL http://www.supplychain.com/purchaseorder.html
     Both successful and invalid read request are logged using
     Obligation.

     If users have a different domain name other than
     @javadotnetinterop.com, this policy will deny access.
     If users with a domain name @javadotnetinterop.com who
     are NOT trading partners this policy also deny their access.

     This policy illustrates use of "Condition" within a
     "Target" element to apply constraints to the read access
     for the requester who are Administrator only. It also
     provides an example of "Obligation"
     to log successful read and log invalid access.
  </Description>

  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId=
            "urn:oasis:names:tc:xacml:2.0:function:rfc822Name-match">
          <AttributeValue
             DataType=
             "http://www.w3.org/2001/XMLSchema#string">
              javadotnetinterop.com
          </AttributeValue>
          <SubjectAttributeDesignator
              DataType=
              "urn:oasis:names:tc:xacml:2.0:data-type:rfc822Name"
              AttributeId=
              "urn:oasis:names:tc:xacml:2.0:subject:subject-id"/>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
        <Resource>
          <ResourceMatch
             MatchId=
```

Fisher_CH13 3/24/06 5:05 PM Page 431

```
           "urn:oasis:names:tc:xacml:2.0:function:anyURI-equal">
          <AttributeValue
             DataType=
             "http://www.w3.org/2001/XMLSchema#anyURI">
             http://www.supplychain.com/purchaseorder.html
          </AttributeValue>
          <ResourceAttributeDesignator
             DataType=
             "http://www.w3.org/2001/XMLSchema#anyURI"
             AttributeId=
             "urn:oasis:names:tc:xacml:2.0:resource:resource-id"/>
        </ResourceMatch>
      </Resource>
  </Resources>
  <Actions>
    <AnyAction/>
  </Actions>
</Target>

<Rule RuleId="ExecuteRule" Effect="Permit">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch
           MatchId=
           "urn:oasis:names:tc:xacml:2.0:function:string-equal">
           <AttributeValue
               DataType="http://www.w3.org/2001/XMLSchema#string">
               execute
            </AttributeValue>
          <ActionAttributeDesignator
             DataType=
             "http://www.w3.org/2001/XMLSchema#string"
             AttributeId=
             "urn:oasis:names:tc:xacml:2.0:action:action-id"/>
```

continues

Listing 13-4 (continued)

```
         </ActionMatch>
       </Action>
     </Actions>
   </Target>
         <Condition
            FunctionId=
            "urn:oasis:names:tc:xacml:2.0:function:string-equal">
       <Apply FunctionId=
       "urn:oasis:names:tc:xacml:2.0:function:string-one-and-only">
         <SubjectAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="group"/>
       </Apply>
       <AttributeValue
          DataType=
          "http://www.w3.org/2001/XMLSchema#string">
          tradingPartner
       </AttributeValue>
     </Condition>
 </Rule>

 <Rule RuleId="DenyOtherActions" Effect="Deny"/>

 <Obligations>
   <Obligation
      ObligationId="LogSuccessfulExecute"
      FulfillOn="Permit">
     <AttributeAssignment
        AttributeId="user"
        DataType=
        "http://www.w3.org/2001/XMLSchema#anyURI">
          urn:oasis:names:tc:xacml:2.0:subject:subject-id
     </AttributeAssignment>
     <AttributeAssignment
        AttributeId="resource"
        DataType="http://www.w3.org/2001/XMLSchema#anyURI">
        urn:oasis:names:tc:xacml:2.0:resource:resource-id
     </AttributeAssignment>
   </Obligation>
   <Obligation
```

```
       ObligationId="LogInvalidAccess"
       FulfillOn="Deny">
      <AttributeAssignment
         AttributeId="user"
         DataType="http://www.w3.org/2001/XMLSchema#anyURI">
         urn:oasis:names:tc:xacml:2.0:subject:subject-id
      </AttributeAssignment>
      <AttributeAssignment
         AttributeId="resource"
         DataType="http://www.w3.org/2001/XMLSchema#anyURI">
         urn:oasis:names:tc:xacml:2.0:resource:resource-id
      </AttributeAssignment>
      <AttributeAssignment
         AttributeId="action"
         DataType="http://www.w3.org/2001/XMLSchema#anyURI">
         urn:oasis:names:tc:xacml:2.0:action:action-id
      </AttributeAssignment>
    </Obligation>
  </Obligations>

</Policy>
```

When applying the XACML security policy using Sun XACML Kit's sample policy engine (SimplePDP), the policy engine shows a positive policy evaluation result, and the service requester is granted access to the URL in question. Refer to Listing 13-5.

Listing 13-5
Evaluating an XACML Policy

```
C:\XACML2\sunxacml-1.2\sample>java SimplePDP request\request.xml
policy\policy.xml
<Response>
  <Result ResourceID=
       "http://www.supplychain.com/purchaseorder.html">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode
         Value="urn:oasis:names:tc:xacml:2.0:status:ok"/>
    </Status>
    <Obligations>
```

continues

Listing 13-5 (continued)

```
      <Obligation
          ObligationId="LogSuccessfulExecute"
          FulfillOn="Permit">
        <AttributeAssignment
            AttributeId="user"
            DataType=
            "http://www.w3.org/2001/XMLSchema#anyURI">
            urn:oasis:names:tc:xacml:2.0:subject:subject-id
        </AttributeAssignment>
        <AttributeAssignment
            AttributeId="resource"
            DataType=
            "http://www.w3.org/2001/XMLSchema#anyURI">
            urn:oasis:names:tc:xacml:2.0:resource:resource-id
        </AttributeAssignment>
      </Obligation>
    </Obligations>
  </Result>
</Response>
```

WS-Policy

Policies are useful in specifying the conditions or assertions regarding interactions between Java and .NET interoperable applications. These policies can be defined for authentication, authorization, quality of protection, quality of service, privacy, reliable messaging, and service-specific options (such as bandwidth guarantee). For Java EE .NET interoperable solutions using Web services, there are two emerging policy-related specifications: WS-Policy and WSPL (Web services policy language).

WS-Policy (Web Services Policy) framework is part of the Web Services roadmap and specifications (a.k.a., WS*) proposed by Microsoft, IBM, VeriSign, and others. It is primarily a policy language that defines polices for Web services. WS-Policy encodes the policy definition in XML using SOAP messages for data exchange. These Web services policies are a collection of "policy alternatives," which are a collection of policy assertions such as authentication scheme, privacy policy, and so forth. This policy framework is a flexible mechanism to define rules for executing Java and .NET applications even though they run on different software platforms and different underlying implementations. Currently, there is a draft JSR 265 specification

Fisher_CH13 3/24/06 5:05 PM Page 435

for Web services policy (refer to www.jcp.org/en/jsr/detail?id=265 &showPrint for details).

Please note that WS-Policy is still not yet an open standard. Some extensions and usage of WS-Policy are now defined in the WS-SecurityPolicy specification. The new OASIS Web Services Secure Exchange (WS-SX) Technical Committee (http://www.oasis-open.org/committees/tc_home .php?wg_abbrev=ws-sx) is working on finalizing a set of specifications based on WS-SecureConversation, WS-SecurityPolicy, and WS-Trust specifications.

Unlike XACML, the WS-Policy specification does not restrict the policy definitions to access control or privacy. WS-Policy can be used to specify the type of security token, digital signature algorithm, and encryption mechanism for a SOAP message (for example, a purchase order message) or even partial contents of a SOAP message (a credit card number, for example). In addition, it can also specify data-privacy or data-confidentiality rules. Nevertheless, WS-Policy does not specify how to discover policies or how to attach a policy to a Web service. It relies on other WS* specifications (for example, WS-PolicyAttachment) to provide full functionality of policy management.

Listing 13-6 shows an example of a WS-Policy that uses Triple DES and RSA OAEP (RSA Optimal Asymmetric Encryption Padding) encryption key algorithms.

Listing 13-6
Sample WS-Policy File

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="microsoft.web.services2"
type="Microsoft.Web.Services2.Configuration.WebServicesConfiguration,
Microsoft.Web.Services2, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
  </configSections>
  <microsoft.web.services2>
    <security>
      <x509 storeLocation="CurrentUser" allowTestRoot="true"
          allowRevocationUrlRetrieval="false" verifyTrust="true" />
      <binarySecurityTokenManager
          valueType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3">
```

continues

Listing 13-6 (continued)

```
        <sessionKeyAlgorithm name="TripleDES" />
        <keyAlgorithm name="RSAOAEP" />
      </binarySecurityTokenManager>
      <securityTokenManager
          xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd"
          qname="wsse:UsernameToken" />
    </security>
    <diagnostics>
      <trace enabled="true" input="InputTrace.webinfo"
          output="OutputTrace.webinfo" />
    </diagnostics>
    <policy>
      <cache name="policyCache.config" />
    </policy>
    <tokenIssuer>
      <autoIssueSecurityContextToken enabled="true" />
    </tokenIssuer>
  </microsoft.web.services2>
</configuration>
```

Under WS-Policy, there is a set of policy assertions for each policy domain. For example, the assertions for use with WS-Security are defined in WS-SecurityPolicy. Each specification or schema to be controlled or managed by WS-Policy requires definition of a new set of assertions.

Under the WS-Policy model, a policy for Web services denotes conditions or assertions regarding the interactions between two Web services endpoints. The service provider exposes a Web services policy for the services they provide. The service requester decides, using the policies, whether it wants to use the service, and if so, the "policy alternative" it chooses to use. In other words, WS-Policy does not have the notion of a Policy Enforcement Point, which enforces policies, and a Policy Decision Point, which also determines policies. It leaves the policy enforcement and decision to the service providers and service requesters.

Web Services Policy Language

WSPL (Web Services Policy Language) is based on XACML and is currently a working draft in the OASIS XACML technical committee. It uses a strict subset of XACML syntax (restricted to Disjunctive Normal Form) and has a different evaluation engine than XACML. XACML evaluates the access-control policies with a given set of attributes and policies, while WSPL determines what the mutually acceptable sets of attributes are when given two policies. For a good introduction on WSPL refer to [Anne3].

WSPL has provided similar functionality to define policies for Web services. WSPL has the semantics of policy and operators, which allow comparison between an attribute of the policy and a value or between two attributes of the policy. The policy syntax also supports rule preference. There are three distinctive features in WSPL. First, it allows policy negotiation, which can merge policies from two sources. Second, policy parameter allows fine-grained parameters such as time of day, cost, or network subnet address to be defined in a policy for Web services. Third, the design of WSPL is flexible enough to support any type of policy by expressing the policy parameters using standard data types and functions.

One main problem WSPL has addressed is the negotiation of policies for Web services. Negotiation is necessary when choices exist or when both parties, Web services consumers and service providers, have preferences, capabilities, or requirements. In addition, it is necessary to automate service discovery and connection related to policies.

WSPL shares similar policy definition capabilities with WS-Policy. Listing 13-7 shows a policy defined in WS-Policy, which specifies the security token usage and type for the Web services. It uses the element, `<ExactlyOne>`, to denote the security token usage.

Listing 13-7
Policy for a Security Token Usage and Type Defined in WS-Policy

```
<wsp:Policy>
   <wsp:ExactlyOne>
      <wsse:SecurityToken>
         <wsse:TokenType>wsse:Kerberosv5TGT
         </wsse:TokenType>
      <wsse:/SecurityToken>
      <wsse:SecurityToken>
         <wsse:TokenType>X509v3
         </wsse:TokenType>
      <wsse:/SecurityToken>
   </wsp:ExactlyOne>
</wsp:Policy>
```

Fisher_CH13 3/24/06 5:05 PM Page 438

Listing 13-8 shows that the same policy can be expressed in WSPL. WSPL translates the policy requirements into two rules. This makes it more descriptive and extensible in the event that security architects and developers need to add more operators or constraints.

Listing 13-8
Sample WSPL Policy Showing Two Rules that Need to Be Satisfied

```
<Policy PolicyId="policy:1" RuleCombiningAlgorithm="&permit-
overrides;">
   <Rule RuleId="rule:1" Effect="Permit">
     Condition FunctionId="&function;string-is-in">
      <AttributeValue DataType="&string;">Kerberosv5TGT</AttributeValue>
      <ResourceAttributeDesignator
          AttributeId="&SecurityToken;"
          DataType="&string;"/>
     </Condition>
   </Rule>

  <Rule RuleId="rule:2" Effect="Permit">
     <Condition FunctionId="&function;string-is-in">
        <AttributeValue
            DataType="&string;">X509v3</AttributeValue>
        <ResourceAttributeDesignator
           AttributeId="&SecurityToken;"
           DataType="&string;"/>
     </Condition>
   </Rule>
</Policy>
```

WS-Policy and WSPL share similar functional features for interoperable Java EE .NET solutions. Anderson has identified a few technical limitations of WS-Policy when compared with WSPL (refer to [Anne2] for details):

- **Negotiation** WS-Policy does not specify a standard merge algorithm or standard way to specify policy negotiation (for example, for merging policies from two sources). Specifications for domain-specific WS-Policy Assertions may describe how to merge or negotiate assertions, but these methods are domain-specific.

Fisher_CH13 3/24/06 5:05 PM Page 439

- **Assertion Comparison** Because there is no standard language for defining Assertions in WS-Policy, there is no standard way to describe requirements such as `minimumPurchaseQuantity > 3000`. Again, specifications for domain-specific WS-Policy Assertions may describe schema elements for such comparisons, but the implementation of these elements must be done on a domain-by-domain basis given there is no standard.
- **Dependency** WS-Policy is designed to depend on extensions. Each extension must be supported by a custom evaluation engine.

Web Single Sign-On Metadata Exchange (SSO MEX)

Both Java and .NET platforms have different approaches in achieving single sign-on. Liberty and SAML protocols are open standards that have wide Java-based implementations. On the .NET side, WS-Federation is used to provide single sign-on functionality. To enable both sides of the single sign-on technologies to interoperate, a new protocol is defined to enable browser-based Web single sign-on between security domains that use different protocols such as Liberty ID-FF and WS-Federation. The Web Single Sign-on Metadata Exchange (Web SSO MEX) primarily specifies a protocol that is independent of the stack and a profile specifying the interoperability between Liberty Identity Federation Framework and WS-Federation. It is not restricted to .NET (or WCF) and Liberty.

The Web SSO MEX protocol defines how a service queries an identity provider for the metadata regarding the identity-federation protocol suites supported by that identity provider. The Web Single Sign-on interoperability profile further describes how the Web SSO MEX protocol is used to enable interoperability between two particular protocol stacks: Liberty's ID-FF 1.2 browser POST profile and WS-Federation Passive Requestor Interoperability profile.

Technology Challenges

A service provider's identity provider may be supporting multiple identity providers, such as Liberty-based and WS-Federation-based identity federation solutions. Both identity management solutions have very different designs and implementations for account federation. For example, Liberty federates different accounts via identity mapping using opaque identifiers, while WS-Federation federates accounts via identity mapping using the

Pseudonym Service. Handling of security tokens can be designed and implemented in different ways as well. For example, Liberty extends SAML assertions for communicating authentication and authorization security tokens between security providers, while WS-Federation uses X.509v3 and Kerberos profiles from the WS-Security specification. For details on the differences between Liberty and WS-Federation, please refer to [LibertyWSFed].

Use Case Scenario

A sample use case scenario (refer to Figure 13-4) would be a Web supply chain portal that supports both Liberty-based and WS-Federation-based identity federation infrastructures. A buyer browses through the online catalog and places purchase orders with two different suppliers. One supplier uses Liberty identity federation framework, denoted in the circles in Figure 13-4, for their supply chain system, and the other supplier uses the WS-Federation identity federation protocol for their order management system, denoted in the triangles in the figure.

Using the Web SSO MEX protocol, the Web portal, acting as the Identity Provider in this scenario, can identify which single sign-on protocol to work with. Each identity federation infrastructure manages its own security token information (such as single sign-on token, SAML assertion, session information, and authentication information). However, there is no data conversion, data exchange, or security token mapping between the two single sign-on systems. Many security vendors are creating products for security token mapping soon. For example, Sun Java System Access Manager (http://www.sun.com/software/products/access_mgr/) now can provide single sign-on token information between the two identity federation infrastructures.

Fisher_CH13 3/24/06 5:05 PM Page 441

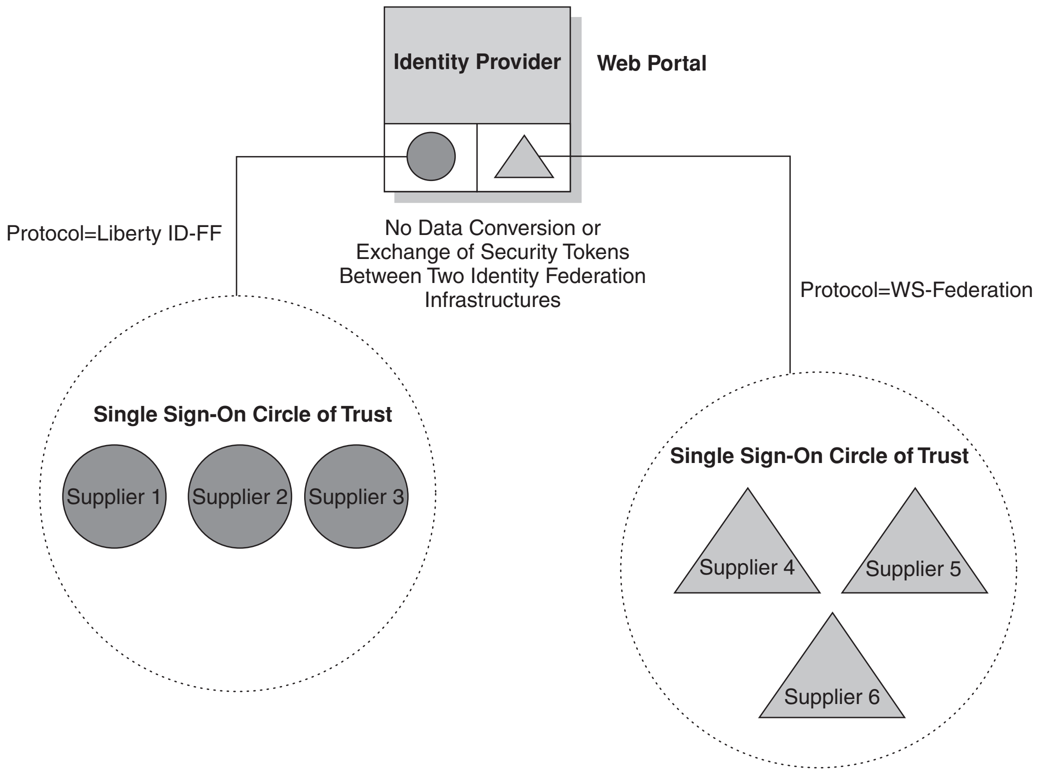

Identity Provider
Web Portal
Protocol=Liberty ID-FF
No Data Conversion or Exchange of Security Tokens Between Two Identity Federation Infrastructures
Protocol=WS-Federation
Single Sign-On Circle of Trust
Supplier 1
Supplier 2
Supplier 3
Single Sign-On Circle of Trust
Supplier 4
Supplier 5
Supplier 6

Figure 13-4
Web portal use case scenario

Figure 13-5 depicts a sequence diagram for the message flow between the client (buyer), the service provider (supplier), and the identity provider (Web portal):

- Upon successful authentication, the client selects the Target Service from the Web portal.
- The client indicates its Identity Provider to the Target Service (Step 1).
- The Target Service formulates the Identity Provider (Step 2) and issues a request for the supported identity federation protocol from the Identity Provider (Step 3).
- The Identity Provider returns the protocol suite document to the Target Service.
- The Target Service determines the appropriate protocol suite (Step 4).

Fisher_CH13 3/24/06 5:05 PM Page 442

- The Target Service begins to exchange documents or messages using the common protocol suite-based single sign-on operations (Step 5). In this step, multiple single sign-on operations are required, and these operations, such as authentication and authentication assertions, vary according to the single sign-on protocols used.
- The Target Service is now able to provide business service based on the user identity (Step 6).

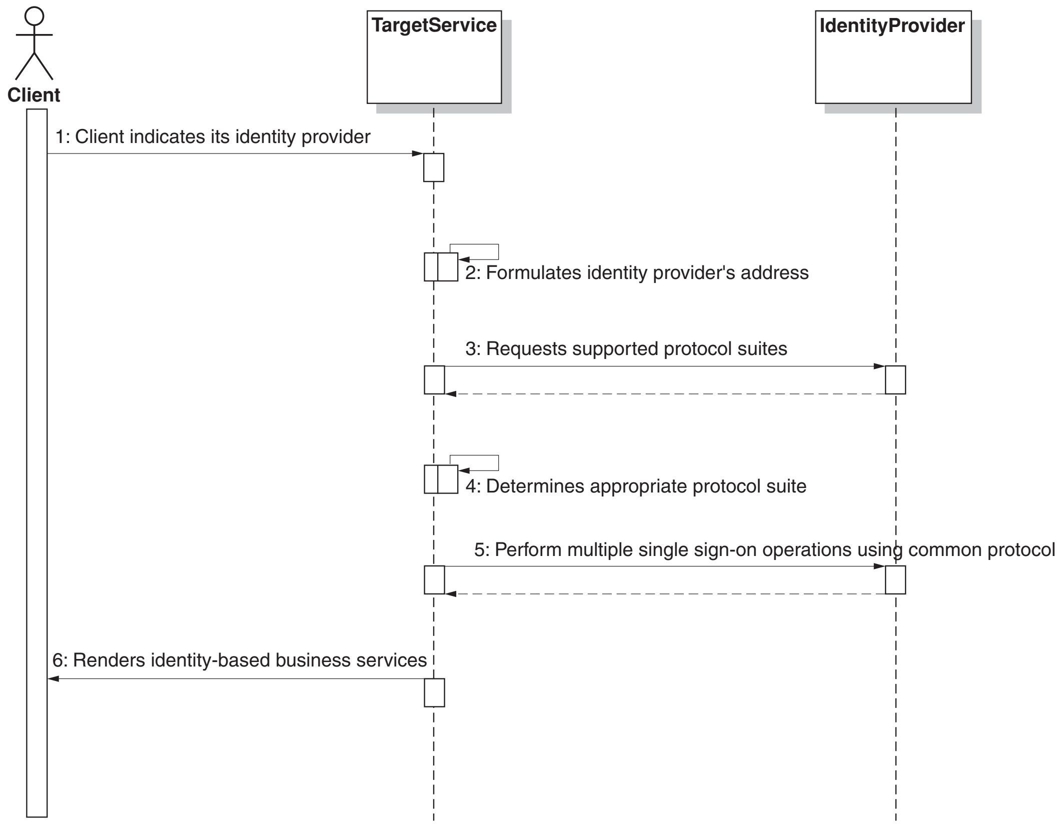

Client
TargetService
IdentityProvider
1: Client indicates its identity provider
2: Formulates identity provider's address
3: Requests supported protocol suites
4: Determines appropriate protocol suite
5: Perform multiple single sign-on operations using common protocol
6: Renders identity-based business services

Figure 13-5
Web SSO sequence diagram

The service provider indicates an identity provider by any of the following four methods:

- Special header in the HTTP request (EPR-base64)
- Query string parameter in the URL
- Custom mechanism (such as mapping table, user prompt)
- EPR constructed using DNS name for the requester (client)

Sample Web SSO MEX Message

Listing 13-9 depicts a request message for Web SSO.

Listing 13-9
Sample Web SSO MEX Message

```
<s12:Header>
   <wsa:Action>
   http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Request
   </wsa:Action>
   <wsa:MessageID>
   uuid:53daedfc-4c3c-38b9-ba46-2480caee43e9
   </wsa:MessageID>
   <wsa:ReplyTo>
      <wsa:Address>
      http://client.javadotnetinterop.com/Endpoint
      </wsa:Address>
   </wsa:ReplyTo>
   <wsa:To>http://service.sun.com/IdentityProvider</wsa:To>
   <ssi:SsiProtocolSuiteHandler/>
</s12:Header>
<s12:Body>
   <wsx:GetMetadata>
      <wsx:Dialect>
      http://schemas.xmlsoap.org/ws/2005/04/SsiSuites
      </wsx:Dialect>
   </wsx:GetMetadata>
</s12:Body>
```

Fisher_CH13 3/24/06 5:05 PM Page 444

Listing 13-10 shows a response message to the MEX request message.

Listing 13-10
Sample Web SSO MEX Reply

```
<s12:Header>
   <wsa:Action>
   http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Response
   </wsa:Action>
   <wsa:MessageID>
   uuid:73d7edfc-5c3c-49b9-ba46-2481caee4177
   </wsa:MessageID>
   <wsa:RelatesTo>
   uuid:73d7edfc-5c3c-49b9-ba46-2480caee43e9
   </wsa:RelatesTo>
   <wsa:To>http://client.javadotnetinterop.com/MyEndpoint</wsa:To>
</s12:Header>
<s12:Body>
   <wsx:Metadata>
      <wsx:MetadataSection
      Dialect='http://schemas.xmlsoap.org/ws/2005/04/SsiSuites' >
         <wsp:ExactlyOne>
         ...
         </wsp:ExactlyOne>
      </wsx:MetadataSection>
   </wsx:Metadata>
</s12:Body>
```

Secure Object Handler Strategy

Problems

It is not uncommon for developers to embed the security processing logic (such as verifying the access rights of the requester and signing the message) in the application code. Such practice allows easy testing of both business functionality and security requirements, without dependency on external program units that provide security functions. This becomes convenient when building Java EE .NET interoperable applications where each platform

Fisher_CH13 3/24/06 5:05 PM Page 445

has its own security processing requirements. However, when the business application grows larger in scale or becomes highly distributed, the maintenance effort and support to manage a change in the security processing logic is enormous. For example, a change in the message digest algorithm or in the access rights processing logic may require considerable program changes in each application program, recompilation and subsequent retesting.

Handlers are processes that can take the service request from the requester's object and process it. For example, a handler can be customized to digitally sign a SOAP message when a service request is created. Thus there is no need to embed the digital signing processing logic in the service requester's application. In essence, handlers are internal representations of actions when an event is triggered or a service request is received. When applied to Java EE .NET interoperability, handlers can be used to execute security processing actions such as digital signing of the SOAP messages, executing the access rights control for a service request, or creating an audit event for the Java EE .NET interoperability actions. Using a secure object handler can decouple the security process from the business processing logic so that the Java EE .NET interoperability solution can be scalable and more manageable.

Solution

The Secure Object Handler strategy applies to both synchronous, asynchronous and bridge interoperability strategies. In essence, the secure object handler intercepts the service request and adds custom processing logic, such as adding a digital signature, to the incoming business data object. Using the object handler strategy, developers do not need to rewrite the application processing logic because they can modify existing handlers directly or add new handlers, or chained handlers.

With synchronous or asynchronous integration, the Secure Object Handler can take the service requester's message or object, and apply the handler action, such as digital signing, encryption/decryption, or tracking the security actions for security audit or compliance reasons. Using a bridge interoperability strategy via, for example, an Enterprise Service Bus, the Secure Object Handler can act as a "service engine" or transformer process for the inbound or outbound service provider.

Although using a handler is a neat strategy, not all interoperability technologies can support the same implementation approach of creating a Secure Object Handler. The following sections discuss different types of technology options for each interoperability strategy that can implement a Secure Object Handler.

Fisher_CH13 3/24/06 5:05 PM Page 446

Synchronous Integration

Synchronous integration enables a Java client to interoperate with a .NET application in a synchronous mode, and vice versa for a .NET client with a Java application. In the synchronous communication between the service requester (client) and the service provider (application), the client waits in a "blocked state," in which no processing or communication with other processes are made until the service provider completes the service request.

.NET Remoting and RPC-style Web services are examples of synchronous integration technologies. .NET implemented in C#, for example, implements the notion of a handler programmatically at the application code level. The client needs to invoke the handler explicitly. If a developer needs to change the handler or update the processing logic in the handler, he or she needs to modify the application codes, recompile, test, and deploy them again.

Listing 13-11 shows a sample secure object handler code that processes a group of service requests (in string array). If the incoming service request contains the string `password`, then the client invokes the handler `Encrypting` to encrypt the service request. If the incoming service request contains the string `creditCard` or `cashPayment`, then the client invokes the handler `DigitalSigning` to perform digital signature. If the service request contains the string `news`, the client invokes the handler `SecureLogging` to log the service request for audit control.

Listing 13-11
Sample Secure Object Handler in .NET Using Synchronous Integration

```
using System;

/// <summary>
///     Abstract class definition for handler
/// </summary>
abstract class Handler {

   protected Handler succeedBy;

/// <param name="succeedBy" type="Handler">
///set next successor to the current method</param>
   public void SetSuccessor(Handler succeedBy) {
     this.succeedBy = succeedBy;
   }
```

```
/// <param name="serviceRequest" type="string">
///invoking secure object handler</param>
   abstract public void SecureObjectHandler(string serviceRequest);
}

/// <summary>
///      Encrypting handler will encrypt the data object or message
///      using XML-Encryption and WS-Security
/// </summary>
class Encrypting: Handler {

/// <param name="serviceRequest" type="string">
///invoking secure object handler for Encrypting</param>
   override public void SecureObjectHandler(string serviceRequest) {
       if (serviceRequest == "password")
       //
       // add your encryption processing logic
          Console.WriteLine("'{0}' secure object handler processed for
'{1}'",
             this, serviceRequest);
       else if (succeedBy != null)
          succeedBy.SecureObjectHandler(serviceRequest);
  }
}

/// <summary>
///     Digital Signing handler will digitally sign the data
///     object or message using
///     XML-Signature and WS-Security
/// </summary>

class DigitalSigning : Handler {

/// <param name="serviceRequest" type="string">
///invoking secure object handler DigitalSigning</param>
   override public void SecureObjectHandler(string serviceRequest) {
       if ((serviceRequest == "cashPayment") ||
           (serviceRequest == "creditCard"))
       //
       // add your digital signing processing logic
```

continues

Fisher_CH13 3/24/06 5:05 PM Page 448

Listing 13-11 (continued)

```
         Console.WriteLine("'{0}'
            secure object handler processed for '{1}'",
            this, serviceRequest);
      else if (succeedBy != null)
         succeedBy.SecureObjectHandler(serviceRequest);
     }
}

/// <summary>
///     SecureLogging handler will log the user info and business
///     transaction reference
///     for audit trail and/or compliance
/// </summary>
class SecureLogging : Handler {

/// <param name="serviceRequest" type="string">
///invoking secure object handler SecureLogging</param>
   override public void SecureObjectHandler(string serviceRequest) {
      if (serviceRequest == "news")
      //
      // add your secure logging processing logic
         Console.WriteLine("'{0}' secure object handler processed for
'{1}'",
            this, serviceRequest);
      else if (succeedBy != null )
         succeedBy.SecureObjectHandler(serviceRequest);
     }
}


/// <summary>
///     Client class to invoke a business transaction or service
///     request.
///     The secure object handler will be invoked.
/// </summary>

public class Client {


/// <summary>
///     Main method for Client class
/// </summary>
  public static void Main(string[] args) {

    // Setup Chain of Responsibility
    Handler encrypting = new Encrypting();
    Handler digitalSigning = new DigitalSigning();
    Handler secureLogging = new SecureLogging();
    encrypting.SetSuccessor(digitalSigning);
    digitalSigning.SetSuccessor(secureLogging);

    // Generate and process serviceRequest
    string[] serviceRequests = {"password", "creditCard", "cashPayment",
"news", "others"};

    Console.WriteLine("Secure Object Handler for Synchronous Integration
- Example");
    Console.WriteLine();
    Console.WriteLine();

    foreach (string serviceRequest in serviceRequests)
       encrypting.SecureObjectHandler(serviceRequest);

    Console.Read();
  }
}
```

RPC-style synchronous Web services usually have support of handlers at the container level. For example, Sun Java Web Services Developer Pack (JWSDP) version 1.6 and Apache Axis version 1.2 allow developers to add handlers without modifying the application codes. In JWSDP, developers can specify a configuration file for their server security environment configuration, such as the certificate alias of the digital signature and the name of the handler. Refer to Listing 13-12 for the sample configuration file. The sample handler `SecurityEnvironmentHandler.java` is an implementation of a `CallbackHandler` that provides digital signature and encryption functionality. The application codes do not need to embed any of the security processing logic.

Listing 13-12
Sample Security Environment Configuration File *wsse.xml* in JWSDP

```
<xwss:JAXRPCSecurity
xmlns:xwss="http://java.sun.com/xml/ns/xwss/config">

<xwss:Service>
    <xwss:SecurityConfiguration dumpMessages="true">

      <xwss:Sign>
          <xwss:X509Token certificateAlias="s1as"/>
      </xwss:Sign>

      <xwss:Encrypt>
          <xwss:X509Token certificateAlias="wse2client"/>
      </xwss:Encrypt>

     </xwss:SecurityConfiguration>
 </xwss:Service>

<xwss:SecurityEnvironmentHandler>
    com.sun.xml.wss.sample.SecurityEnvironmentHandler
</xwss:SecurityEnvironmentHandler>

</xwss:JAXRPCSecurity>
```

When building secure object handlers for synchronous integration, developers need to ensure the client is properly authenticated with the server and authorized for the business services prior to invoking the services. This is to ensure that no unauthorized user can invoke the business service. Having these security measures in place addresses the risks of confidentiality, principal spoofing, repudiation, broken authentication, and broken access control.

Asynchronous Integration

Asynchronous integration is loosely coupled interaction between the service requester (client) and the service provider (server). Unlike synchronous integration, asynchronous integration does not require that the client go into a blocked state until the processing of the service request is complete.

Document-style Web service is a common example of asynchronous integration using Web services. It encapsulates the service request or the reply in the form of a message, which can be encrypted or decrypted for confidentiality and digitally signed for non-repudiation. Synchronous integration also uses the same handler mechanism in the Web services container, such as JWSDP or Axis, as discussed earlier, to encapsulate the security processing logic.

When building secure object handlers, asynchronous integration requires similar security requirements, such as authenticating the service requester with the server, as in synchronous integration. The emphasis is on securing the message instead of securing the service requester or the service provider. The sender of the message can be authenticated prior to sending the message. Alternatively the message can be authenticated, digitally signed with a valid certificate for example, and verified intact or not being modified upon receipt to ensure that it is a valid message. Having these security measures in place would address the risks of message alteration, message replay, and repudiation.

Integration Using Enterprise Service Bus

Java EE .NET interoperability using a bridge or an Enterprise Service Bus (ESB) supports both synchronous and asynchronous integration strategies. The bridge or ESB acts as an intermediary between the Java and .NET platform. The security requirements of both synchronous and asynchronous integration strategies discussed earlier also apply to the ESB integration strategy.

ESB has recently become a common example of interoperability in a Service Oriented Architecture (SOA) environment probably because of the multi-messaging protocol and agility in integrating heterogeneous platforms. Not all ESB products support the implementation of a handler. In the example of Mule ESB (http://mule.codehaus.org), Mule allows handlers to be added to an inbound or outbound process. Listing 13-13 shows an example of a Mule ESB configuration where the tag `inboundTransformer` can specify a secure object handler. The object `SecureObjectHandler` is a Java class in the current class path that can be programmed to perform security processing such as digital signature for the incoming messages or service requests. Similarly, a secure object handler can be added under the tag `outboundTransformer` to embed any security processing for outbound messages or reply. For example, all replies to the service requests need to be encrypted and digitally signed using a Secure Object Handler.

Listing 13-13
Sample Mule ESB Configuration to Specify a Secure Object Handler

```
<?xml version="1.0" encoding="UTF-8"?>
...
<mule-configuration id="sampleProperties" version="1.0">
   <connector name="SystemStreamConnector"
className="org.mule.providers.stream.SystemStreamConnector">
        <properties>
         ...
        </properties>
    </connector>

    <model name="MuleClient">
        <mule-descriptor name="ESBClient"
            inboundEndpoint="stream://System.in"
            inboundTransformer="SecureObjectHandler"
            outboundEndpoint="vm://mule/receive"
            implementation="com.sun.esb.samples.MuleClient">
       </mule-descriptor>
    </model>
</mule-configuration>
```

The technology details for creating a handler vary depending on the ESB product used. It is important to ensure that the client using the ESB is properly authenticated. Moreover, the ESB product adopted should allow adding or modifying Secure Object Handlers without changing the application processing logic.

Creating Handlers

There is no common or standard way to implement handlers. For example, developers can use a callback handler design approach to include the security processing logic. For instance, JWSDP has a sample handler `SecurityEnvironmentHandler.java` in the samples directory. Each platform or product may have different mechanisms to implement and customize Secure Object Handlers. The following design considerations can apply regardless of the platform chosen:

Fisher_CH13 3/24/06 5:05 PM Page 453

- Decouple the security processing logic from the business processing logic. Factor the common security processing logic, such as digital signing, into a Secure Object Handler.
- Use the application server infrastructure, such as JWSDP with Sun Java System Application Server, to support chained handlers if possible. Refer to earlier sample configuration files.
- Use a policy framework to implement handlers because policy is declarative and is easier to make changes to when compared to implementing handlers programmatically. Listing 13-14 shows an example of specifying the handler in the application configuration file `policyCache.config`, which can be generated by using the built-in WSE policy editor.

Listing 13-14
Sample *policyCache.config* File

```
<?xml version="1.0" encoding="utf-8"?>
<policyDocument
    xmlns="http://schemas.microsoft.com/wse/2003/06/Policy">

  <mappings
      xmlns:wse="http://schemas.microsoft.com/wse/2003/06/Policy">
    <!--The following policy describes the policy requirements for the
service: http://localhost:8080/OrderService/OrderService .-->
    <endpoint uri="http://localhost:8080/OrderService/OrderService">
      <defaultOperation>
        <request policy="#Sign-X.509" />
        <response policy="" />
        <fault policy="" />
      </defaultOperation>
    </endpoint>
  </mappings>

  <policies
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
      xmlns:wssp="http://schemas.xmlsoap.org/ws/2002/12/secext"
      xmlns:wse="http://schemas.microsoft.com/wse/2003/06/Policy"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
```

continues

Listing 13-14 (continued)

```
wss-wssecurity-secext-1.0.xsd"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">

    <wsp:Policy
        wsu:Id="Sign-X.509">
      <!--MessagePredicate is used to require headers. This assertion
should be used along with the Integrity assertion when the presence of
the signed element is required. NOTE: this assertion does not do
anything for enforcement (send-side) policy.-->

      <wsp:MessagePredicate
          wsp:Usage="wsp:Required"
          Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part">
          wsp:Body()
          wsp:Header(wsa:To)
          wsp:Header(wsa:Action)
          wsp:Header(wsa:MessageID)
          wse:Timestamp()
      </wsp:MessagePredicate>
      <!--The Integrity assertion is used to ensure that the message is
signed with X.509. Many Web services will also use the token for
authorization, such as by using the <wse:Role> claim or specific X.509
claims.-->
      <wssp:Integrity
          wsp:Usage="wsp:Required">
        <wssp:TokenInfo>
          <!--The SecurityToken element within the TokenInfo element
describes which token type must be used for Signing.-->
          <wssp:SecurityToken>
            <wssp:TokenType>
                http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3
            </wssp:TokenType>
            <wssp:TokenIssuer>
                CN=Root Agency
            </wssp:TokenIssuer>

            <wssp:Claims>
              <!--By specifying the SubjectName claim, the policy system
can look for a certificate with this subject name in the certificate
store indicated in the application's configuration, such as LocalMachine
```

```
or CurrentUser. The WSE X.509 Certificate Tool is useful for finding the
correct values for this field.-->
             <wssp:SubjectName
                 MatchType="wssp:Exact">
                 CN=WSE2QuickStartClient
             </wssp:SubjectName>
             <wssp:X509Extension
                 OID="2.5.29.14"
                 MatchType="wssp:Exact">gBfo0147lM6cKnTbbMSuMVvmFY4=
             </wssp:X509Extension>
           </wssp:Claims>

         </wssp:SecurityToken>
       </wssp:TokenInfo>

       <wssp:MessageParts
           Dialect=
              "http://schemas.xmlsoap.org/2002/12/wsse#part">
            wsp:Body() wsp:Header(wsa:Action)
            wsp:Header(wsa:FaultTo)
            wsp:Header(wsa:From)
            wsp:Header(wsa:MessageID)
            wsp:Header(wsa:RelatesTo)
            wsp:Header(wsa:ReplyTo)
            wsp:Header(wsa:To)
            wse:Timestamp()
       </wssp:MessageParts>

     </wssp:Integrity>
   </wsp:Policy>
  </policies>
</policyDocument>
```

WSE 2.0 provides a configuration editor that reads the input from the `app.config` or `Web.config` file and generates the policy cache file in Listing 13-14. (Refer to the "Example" section later in the chapter for more details.) The policy Sign-X.509 and the handler details are referenced in the `app.config` or `Web.config` file. If the incoming SOAP request with the element `<wsse:BinarySecurityToken>` does not contain the appropriate binary security token that meets the policy defined here, an exception is thrown.

Fisher_CH13 3/24/06 5:05 PM Page 456

If a developer wants to a add new handler, he can write a custom policy assertion handler that is derived from the `Microsoft.Web.Services.Policy.PolicyAssertion` class and register it in the `app.config` or `Web.config` file. Refer to Listing 13-15 for an example.

Listing 13-15
Adding a Custom Handler in the *app.config* or *Web.config* Configuration Files

```
using Microsoft.Web.Services.Policy;
public class MySecureObjectHandler : PolicyAssertion {
   // add your customized security processing logic here
   // to override the member functions
}


In the app.config or Web.config configuration file, you need to add an
assertion element to the policy section.  This assertion element maps a
policy assertion element name (such as wsp:MessagePredicate) and
registers it in the configuration file.

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  …
  </configSections>
  <microsoft.web.services>
   <policy>
      <receive>
         <cache name="policyCache.xml"/>
      </receive>
      <assertion name="wsp:MessagePredicate"
                 type="SecureObjectHandler"
                 xmlns:wsp="…" />
   </policy>
  </microsoft.web.services>
   …
</configuration>
```

In JWSDP, a handler can be customized to extend a callback handler. Listing 13-16 shows a sample template where you can customize your handlers under the public method "handle." For details, please refer to the JWSDP API documentation to implement a callback handler.

Listing 13-16
Sample *SecurityEnvironmentHandler* Callback Handler Template in JWSDP

```
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import com.sun.xml.wss.impl.callback.CertificateValidationCallback;
import com.sun.xml.wss.impl.callback.DecryptionKeyCallback;
import com.sun.xml.wss.impl.callback.EncryptionKeyCallback;
import com.sun.xml.wss.impl.callback.PasswordCallback;
import com.sun.xml.wss.impl.callback.PasswordValidationCallback;
import com.sun.xml.wss.impl.callback.SignatureKeyCallback;
import com.sun.xml.wss.impl.callback.SignatureVerificationKeyCallback;
import com.sun.xml.wss.impl.callback.UsernameCallback;
import com.sun.org.apache.xml.security.utils.RFC2253Parser;
// insert your additional Java class libraries...
…

public class SecureObjectHandler implements CallbackHandler {
    // ...

    public SecureObjectHandler() throws Exception {
        // define your constructor details
    }

    public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
        for (int i=0; i < callbacks.length; i++) {

            if (callbacks[i] instanceof PasswordValidationCallback) {
                // implement your own password validation processing
                // logic
            } else if (callbacks[i] instanceof
SignatureVerificationKeyCallback) {
                // implement your own digital signature verification
                // processing logic
```

continues

Fisher_CH13 3/24/06 5:05 PM Page 458

Listing 13-16 (continued)

```
            } else if (callbacks[i] instanceof SignatureKeyCallback) {
                // implement your own signature key processing logic
            } else if (callbacks[i] instanceof DecryptionKeyCallback) {
                // implement your own decryption key processing logic
            } else if (callbacks[i] instanceof EncryptionKeyCallback) {
                // implement your own encryption key processing logic
            } else if (callbacks[i] instanceof
CertificateValidationCallback) {
                // implement your own digital certificate validation
                //  processing logic
            } else {
                throw unsupported;
            }
        }
    }
    // ...
}
```

It's best to use known interoperable encryption key or session algorithms. For example, RSA Optimal Asymmetric Encryption Padding (RSAOREA) and Triple DES are interoperable between .NET and JWSDP.

Example

This example illustrates a .NET client (Retailer) placing a purchase order with the service provider (Manufacturer). The .NET client invokes a purchase order Web service from a Java application service with the details of the account number, SKU number, supplier item number, and quantity. Upon completion of the purchase order submission, the service provider returns a receipt number.

The .NET client uses the installed X.509 certificate to sign the service request and encrypt the business data. The JWSDP server validates the message digest and decrypts the business data. This example demonstrates the interoperability of the digital signature and the encryption/decryption between the .NET client and the JWSDP server using Web services security.

Gates provides a comprehensive example for Java EE .NET interoperability. See [SGuest] for more details.

Fisher_CH13 3/24/06 5:05 PM Page 459

Assumptions

The following software components need to be installed and configured on the same machine:

- **Java Web Services Developer Pack (JWSDP) version 1.5 (http://java.sun.com/webservices/downloads/webservicespack.html)** A Tomcat container *customized* for JWSDP (http://java.sun.com/webservices/containers/tomcat_for_JWSDP_1_5.html) also must be installed. Please note that the standard Apache Tomcat server is not compatible here. In this example, the product is installed under `C:\Tools\Tomcat`. Please refer to the product installation documentation for details.
- **Web Services Enhancement 2.0 SP1 (http://msdn.microsoft.com/webservices/)** In this example, the product is installed under the default directory `C:\Program Files\Microsoft WSE\v2.0`. Please refer to the product installation documentation for details.
- **.NET Framework SDK version 1.1 (http://msdn.microsoft.com/netframework/downloads/framework1_1)** In this example, the product is installed under the default directory `C:\Program Files\Microsoft.NET\SDK\v1.1`. Please refer to the product installation documentation for details.
- **Java 2 Standard Edition SDK version 1.5.0_03 http://java.sun.com/j2se/1.5.0/download.jsp)**

Prerequisites

To illustrate security interoperability using Web services security, this example requires the following:

- .NET client installs a sample X.509 certificate entitled WSE2QuickStartClient (`"Client Private.pfx"`) from the WSE 2.0 (under `C:\Program Files\Microsoft WSE\v2.0\Samples\Sample Test Certificates`). The public key of the client certificate also must be imported into the Java Keystore via Microsoft Management Console (`mmc.exe`) as the name "`wse2client.cer`" under `C:\Tools\JWSDP\xws-security\etc`.
- Using the `WSE2QuickStartClient` certificate, the root agency must be exported to a file wse2ca.cer under `C:\Tools\JWSDP\xws-security\etc`.

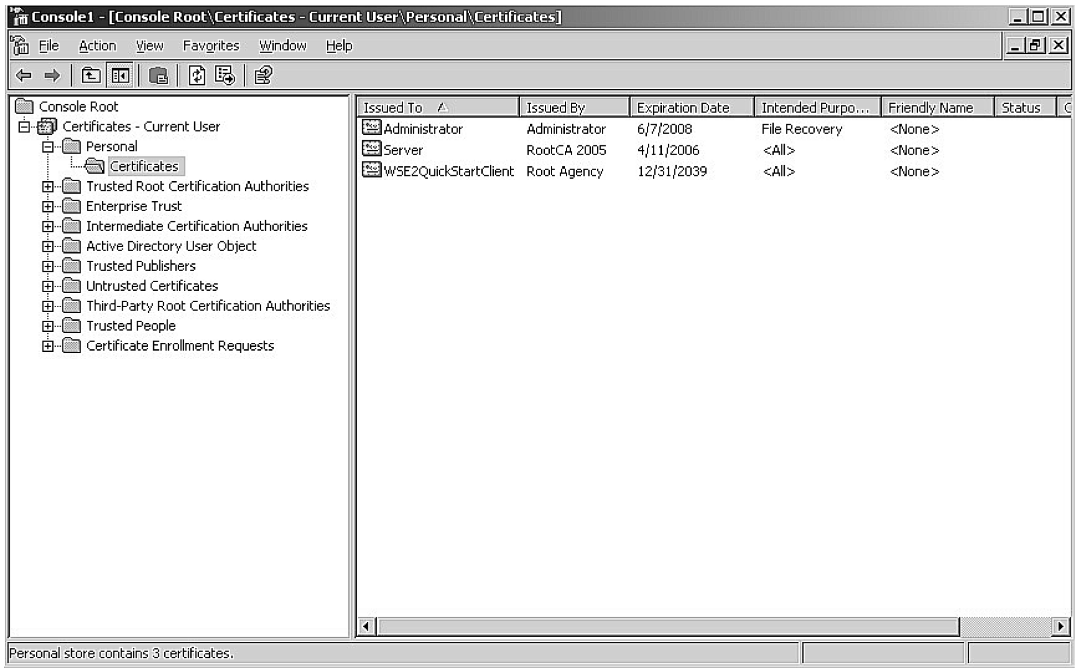
- JWSDP server uses the sample X.509 certificates from the JWSDP distribution. Export the JWSDP server (`server.cer`) and root agency (`ca.cer`) certificates from the keystores `server-keystore.jks` and `server-truststore.jks` under the same directory `C:\Tools\JWSDP\xws-security\etc`).
- Install a JCE RSA provider. In this example, the latest Bouncy Castle JCE provider (BCP) for Java 2 SE SDK (aka JDK) version 1.5 (www.bouncycastle.org/latest_releases.html) is used. Ensure that the BCP version (such as BCP for JDK 1.5) matches the same JDK version number (such as JDK 1.5) for compatibility.
- The .NET client is configured with a WS-Security setting of XML signature and encryption using the built-in WSE Configuration Editor.
- The JWSDP server is configured with the default `SecurityEnvironmentHandler` to provide XML signature and decryption/encryption.

Procedures

1. **Install .NET Client Certificate.**
   Go to `C:\Program Files\Microsoft WSE\v2.0\Samples\Sample Test Certificates` and double-click the file `Client Private.pfx`. This should start the Certificate Import Wizard to install the certificate and private key. The Certificate Import Wizard prompts you for the password (which is *wse2qs*). Select the target folder location as "`Personal`."

   To verify the import operation, you can open Microsoft Management Console (`mmc.exe`) to browse the entry `WSE2QuickStartClient` from the Personal folder. This can be done by running the command `mmc.exe` from the Start/Run of the Windows environment. Then select File/Add/Remove Snap-in…/Add/Certificates/My User Account. Microsoft Management Console (MMC) should then show a hierarchy of certificates folders under "Console Root"/"Certificates – Current User" (refer to Figure 13-6).

Fisher_CH13 3/24/06 5:05 PM Page 461


Console1 - [Console Root\Certificates - Current User\Personal\Certificates]
File
Action
View
Favorites
Window
Help
Console Root
Certificates - Current User
Personal
Certificates
Trusted Root Certification Authorities
Enterprise Trust
Intermediate Certification Authorities
Active Directory User Object
Trusted Publishers
Untrusted Certificates
Third-Party Root Certification Authorities
Trusted People
Certificate Enrollment Requests

Issued To	Issued By	Expiration Date	Intended Purpo...	Friendly Name	Status
Administrator	Administrator	6/7/2008	File Recovery	<None>	
Server	RootCA 2005	4/11/2006	<All>	<None>	
WSE2QuickStartClient	Root Agency	12/31/2039	<All>	<None>	

Personal store contains 3 certificates.

Figure 13-6
Verifying client certificate that is imported

Now you can export the public part of the `WSE2QuickStartClient` certificate to the Java keystore under `C:\Tools\Tomcat\xws-security\etc`. Open the Microsoft Management Console (as depicted earlier in Figure 13-6). From the MMC, right-click the `WSE2QuickStartClient` entry and select "All Tasks | Export" to export the private key in "Base 64 CER" export type under the file name `C:\Tools\Tomcat\xws-security\etc\wse2client.cer`.

After that, double-click the `WSE2QuickStartClient` certificate again, select the Certification Path tab and double-click the `Root Agency` certificate. Then click the Details tab and select the Copy to File button to save the root agency certificate as `C:\Tools\Tomcat\xws-security\etc\wse2ca.cer`.

To import these client certificates into Java keystore `server-trust-store.jks` so that the JWSDP server application can process the XML signature and decryption using the public key, invoke the following commands:

```
keytool –import –file wse2client.cer –alias wse2client
–keystore server-truststore.jks

keytool -import -file wse2ca.cer -alias wse2ca -key-
store server-truststore.jks
```

You can verify the import operation by the following commands (refer to Listing 13-17):

Listing 13-17
Verifying Import of .NET Client Certificate

```
C:\Tools\Tomcat\xws-security\etc>keytool -printcert -file
wse2client.cer
Owner: CN=WSE2QuickStartClient
Issuer: CN=Root Agency
Serial number: -3abb68e8ca769e74b1998659471cf56e
Valid from: Tue Jul 08 11:47:59 PDT 2003 until: Sat Dec 31 15:59:59 PST
2039
Certificate fingerprints:
        MD5:  72:52:48:7C:00:45:53:94:38:BE:47:5B:15:00:80:37
        SHA1:
CA:76:01:38:1B:45:78:50:2B:62:B8:80:98:25:66:4F:1E:78:DF:A2

C:\Tools\Tomcat\xws-security\etc>keytool -printcert -file wse2ca.cer
Owner: CN=Root Agency
Issuer: CN=Root Agency
Serial number: 6376c00aa00648a11cfb8d4aa5c35f4
Valid from: Tue May 28 15:02:59 PDT 1996 until: Sat Dec 31 15:59:59 PST
2039
Certificate fingerprints:
        MD5:  C0:A7:23:F0:DA:35:02:6B:21:ED:B1:75:97:F1:D4:70
        SHA1:
FE:E4:49:EE:0E:39:65:A5:24:6F:00:0E:87:FD:E2:A0:65:FD:89:D4
```

2. **Install JWSDP Server Certificate.**
   Go to C:\Tools\Tomcat\xws-security\etc and run the following commands to export the sample JWSDP server certificates:

   ```
   keytool –export –file server.cer –alias s1as –keystore
   server-keystore.jks

   keytool -export -file ca.cer -alias certificate-
   authority -keystore server-truststore.jks
   ```

   To verify the export operation, you can use the following commands (refer to Listing 13-18):

Listing 13-18
Verifying Export of JWSDP Server Certificate

```
C:\Tools\Tomcat\xws-security\etc>keytool -printcert -file server.cer
Owner: CN=Server, OU=JWS, O=SUN, ST=Some-State, C=AU
Issuer: CN=RootCA 2005, OU=JWS, O=SUN, ST=Some-State, C=IN
Serial number: 2
Valid from: Mon Apr 11 22:37:29 PDT 2005 until: Tue Apr 11 22:37:29 PDT
2006
Certificate fingerprints:
        MD5:  5E:F1:DB:6F:66:22:C6:AC:E8:C3:D9:73:35:C7:2C:AC
        SHA1:
F0:67:9A:4E:1C:FD:F5:F0:C7:39:F7:94:08:3A:EF:54:B3:14:71:12

C:\Tools\Tomcat\xws-security\etc>keytool -printcert -file ca.cer
Owner: CN=RootCA 2005, OU=JWS, O=SUN, ST=Some-State, C=IN
Issuer: CN=RootCA 2005, OU=JWS, O=SUN, ST=Some-State, C=IN
Serial number: 0
Valid from: Mon Apr 11 22:28:27 PDT 2005 until: Thu Apr 09 22:28:27 PDT
2015
Certificate fingerprints:
        MD5:  3D:B5:3C:93:F5:65:D5:3D:B5:C5:2E:23:F5:2E:3A:E9
        SHA1:
98:5F:43:96:C0:ED:A5:88:19:DC:D2:1A:2F:8A:5E:0E:44:42:D7:A1
```

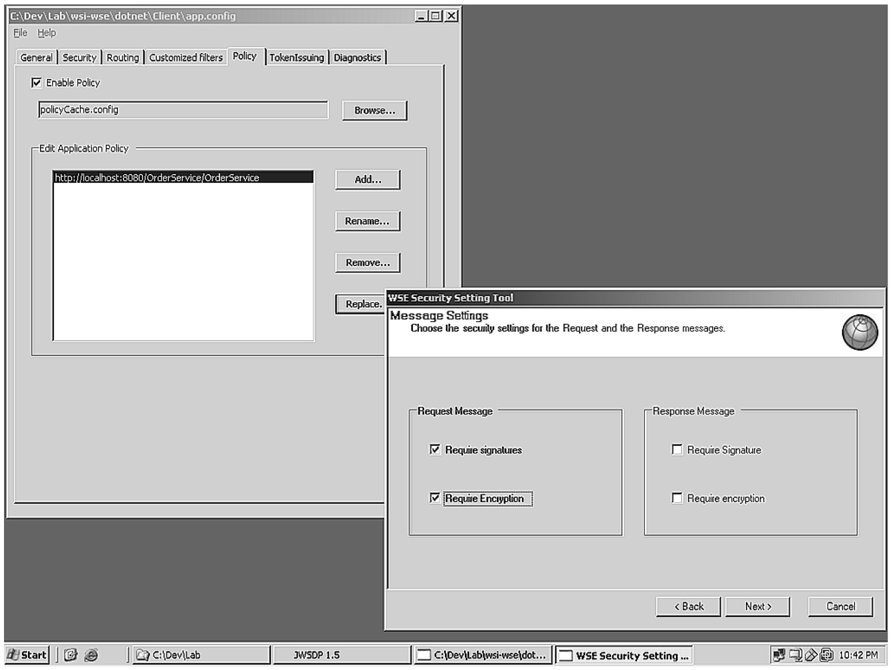
3. **Install JCE RSA Provider.**
   Install the Bouncy Castle JCE provider under the Java 2 SE SDK (JDK) directory. In this example, install under `C:\Tools\JDK15\jre\lib\ext`. Edit your `java.security` policy file under the JDK installation to add the new JCE provider. Please note that the order of the JCE provider may cause compatibility issues. From experience, add the Bouncy Castle JCE provide after the Sun RSA provider (refer to Listing 13-19).

Listing 13-19
Adding Bouncy Castle JCE Provider in *java.security*

```
#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=com.sun.net.ssl.internal.ssl.Provider
security.provider.4=com.sun.crypto.provider.SunJCE
security.provider.5=org.bouncycastle.jce.provider.BouncyCastleProvider
security.provider.6=sun.security.jgss.SunProvider
security.provider.7=com.sun.security.sasl.Provider
```

4. **Configure .NET Client Policy.**
   Open the .NET WSE configuration file `app.config` or `Web.config` using the WSE Configuration Editor (refer to Figure 13-7). This should enable the .NET client side to support XML signature and XML encryption for Web services security.

5. **Configure JWSDP Security Handler.**
   On the JWSDP server side, edit the `wsse.xml` configuration file that JWSDP uses to configure Web services security. The previous shows an example of the configuration file content with both XML signature and XML encryption turned on.


C:\Dev\Lab\wsi-wse\dotnet\Client\app.config

Policy

☑ Enable Policy

policyCache.config

Edit Application Policy

http://localhost:8080/OrderService/OrderService

WSE Security Setting Tool

Message Settings

Choose the security settings for the Request and the Response messages.

Request Message

☑ Require signatures

☑ Require Encryption

Response Message

☐ Require Signature

☐ Require encryption

Figure 13-7
Using WSE Configuration Editor to define XML signature and XML encryption

6. **Compile and Deploy JWSDP Application.**
   After the Web services security features are turned on in the JWSDP server application, you can compile and deploy your server application. This example uses an `ant` script to compile and deploy to JWSDP. For example

```
C:\Dev\Lab\wsi-wse\wsdp\service>ant build-all
Buildfile: build.xml
…
build-all:

BUILD SUCCESSFUL
Total time: 19 seconds
```

7. **Compile and Invoke .NET Client.**

   Once the Web services security features are turned on in the .NET client, you can compile your client application, for example:

```
C:\Dev\Lab\wsi-wse\dotnet\Client>csc /out:Client.exe
*.cs proxy\OrderProxy.cs /r:"c:\program
files\microsoft wse\v2.0\Microsoft.Web.Services2.dll"
Microsoft (R) Visual C# .NET Compiler version
7.10.6310.4
for Microsoft (R) .NET Framework version 1.1.4322
Copyright (C) Microsoft Corporation 2001-2002. All
rights reserved.
```

   In this example, the .NET client application is called `Client.exe`, which is compiled from the existing C# programs in the directory `C:\Dev\Lab\wsi-wse\dotnet\Client`.

Result

The .NET client generates a service request to create a purchase order and renders the following XML message (refer to Listing 13-20). The header contains the key Web Services Security information `<wsse:Security>`, which includes the elements `<wsse:BinarySecurityToken />` and `<ds:Signature />`. The body contains the core business data `<submitOrderResponse />`.

Listing 13-20
Service Request to Create a Purchase Order from a .NET Client

```
<?xml version="1.0" encoding="utf-8"?>
<log>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:ns0="http://wss.samples.microsoft.com"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<env:Header>
   <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd"
```

```
      env:mustUnderstand="1">
      <wsse:BinarySecurityToken
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-utility-1.0.xsd"
          EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-soap-message-security-1.0#Base64Binary"
          ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3"
          wsu:Id="Id-4853980734745059755">
             MIIC8zCCAlygAwIBAgIBAjAN
             …
             Q3u1+58HZRLS97o+vmKy84OE
      </wsse:BinarySecurityToken>

      <ds:Signature
          xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <ds:SignedInfo>
             <ds:CanonicalizationMethod
                 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
             <ds:SignatureMethod
                 Algorithm=
                    "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
             <ds:Reference
                URI="#Id-1069588494982330250">
                <ds:Transforms>
                   <ds:Transform
                       Algorithm=
                          "http://www.w3.org/2001/10/xml-exc-c14n#" />
                </ds:Transforms>

                <ds:DigestMethod
                    Algorithm=
                       "http://www.w3.org/2000/09/xmldsig#sha1" />
                <ds:DigestValue>
                   FD+aZWI8zxgIuqsE9/LbHeGkiuI=
                </ds:DigestValue>
             </ds:Reference>

             <ds:Reference
                 URI="#Id1298574802901223328">
                 <ds:Transforms>
```

continues

Listing 13-20 (continued)

```
                  <ds:Transform
                      Algorithm=
                         "http://www.w3.org/2001/10/xml-exc-c14n#" />
               </ds:Transforms>

               <ds:DigestMethod
                   Algorithm=
                      "http://www.w3.org/2000/09/xmldsig#sha1" />
               <ds:DigestValue>
                   0joA2wLLMj9ZzPMRPEufAv14keI=
               </ds:DigestValue>
           </ds:Reference>
       </ds:SignedInfo>

       <ds:SignatureValue>
           qeYG9xwdSxIMYXi7c4wby5bQkPsqVdOIgi0RbQHW237 …
       </ds:SignatureValue>

       <ds:KeyInfo>
          <wsse:SecurityTokenReference>
             <wsse:Reference
                 URI="#Id-4853980734745059755"
                 ValueType=
                    "http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3" />
          </wsse:SecurityTokenReference>
       </ds:KeyInfo>
    </ds:Signature>

    <wsu:Timestamp
        xmlns:wsu=
           "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
        wsu:Id="Id1298574802901223328">

        <wsu:Created>
            2005-06-28T14:46:42Z
        </wsu:Created>
        <wsu:Expires>
            2005-06-28T14:51:42Z
          </wsu:Expires>
      </wsu:Timestamp>
   </wsse:Security>
</env:Header>

<env:Body
    xmlns:wsu=
       "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
     wsu:Id="Id-1069588494982330250">
   <ns0:submitOrderResponse>
       <result>2233444</result>
   </ns0:submitOrderResponse>
 </env:Body>
</env:Envelope>
</log>
```

The JWSDP server application receives the service request and returns a receipt number in XML (refer to Listing 13-21):

Listing 13-21
Response to the Purchase Order Request by JWSDP Server Application

```
<?xml version="1.0" encoding="utf-8"?>
<log>
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
   xmlns:wsse=
     "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"
   xmlns:wsu=
      "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">

<soap:Header>
   <wsa:Action
      wsu:Id="Id-8f3d108c-3186-4eac-bd30-8584e82fa38d">
   </wsa:Action>
```

continues

Listing 13-21 (continued)

```
   <wsa:MessageID
      wsu:Id="Id-c00ca163-3d7b-4cec-8e7a-791546d41e15">
         uuid:119831ce-58be-4cd1-b70f-77e7ad96552f
   </wsa:MessageID>

   <wsa:ReplyTo
      wsu:Id="Id-42f224be-c812-473e-b243-eea1f2dd7e66">

      <wsa:Address>
http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:Address>
   </wsa:ReplyTo>

   <wsa:To
      wsu:Id=
         "Id-1f1db802-65c0-43d3-9af1-167208ce98e4">
      http://localhost:8080/OrderService/OrderService
   </wsa:To>

   <wsse:Security soap:mustUnderstand="1">
        <wsu:Timestamp wsu:Id=
           "Timestamp-d65bb998-7c43-4efc-808c-2304d8a13d79">
          <wsu:Created>2005-06-28T14:45:59Z</wsu:Created>
          <wsu:Expires>2005-06-28T14:50:59Z</wsu:Expires>
        </wsu:Timestamp>
        <wsse:BinarySecurityToken
           ValueType=
              "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"
           EncodingType=
              "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary"
           xmlns:wsu=
              "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"
           wsu:Id=
              "SecurityToken-1ecd5e58-9ba6-4239-a9fb-
250e15dc3769">MIIBxDCCAW6gAwIBAgIQxUSXFzWJYYtOZnmmuOMKkjANBgkqhkiG9w0BA
Q...F5qkh6sSdWVBY5sT/txBnVJGziyO8DPYdu2fPMER8ajJfl
        </wsse:BinarySecurityToken>
```

Fisher_CH13 3/24/06 5:05 PM Page 471

```
<Signature
    xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <ds:CanonicalizationMethod
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#" />
    <SignatureMethod
        Algorithm=
           "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <Reference
        URI="#Id-8f3d108c-3186-4eac-bd30-8584e82fa38d">
      <Transforms>
        <Transform Algorithm=
          "http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>
          yHqfuGhCwnU/guHlvPy5j8vZDC0=
      </DigestValue>
    </Reference>
    <Reference
        URI="#Id-c00ca163-3d7b-4cec-8e7a-791546d41e15">
      <Transforms>
        <Transform Algorithm=
            "http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod
          Algorithm=
             "http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>
         yM82YC9rZzfb3AWD0iYbNmBT4UM=
      </DigestValue>
    </Reference>
    <Reference
        URI="#Id-42f224be-c812-473e-b243-eea1f2dd7e66">
      <Transforms>
        <Transform Algorithm=
            "http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
      <DigestMethod
```

continues

Listing 13-21 (continued)

```
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>
                5GDwT3MA0YOx4GqugDBtWUyjMnw=
            </DigestValue>
          </Reference>
          <Reference
              URI="#Id-1f1db802-65c0-43d3-9af1-167208ce98e4">
            <Transforms>
              <Transform
                  Algorithm=
                  "http://www.w3.org/2001/10/xml-exc-c14n#" />
            </Transforms>
            <DigestMethod
               Algorithm=
                  "http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>
                nG7Lkpn0W1pMB/BRWiQGltfbyRA=
            </DigestValue>
          </Reference>
          <Reference
              URI=
                 "#Timestamp-d65bb998-7c43-4efc-808c-2304d8a13d79">
            <Transforms>
              <Transform
                  Algorithm=
                     "http://www.w3.org/2001/10/xml-exc-c14n#" />
            </Transforms>
            <DigestMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>
                9udH1FbnHV0eq6BQ8wJUof3V+UA=
            </DigestValue>
          </Reference>
          <Reference
              URI="#Id-42767f08-d88f-4394-a4f3-06bf14b00916">
            <Transforms>
              <Transform
                   Algorithm=
                  "http://www.w3.org/2001/10/xml-exc-c14n#" />
            </Transforms>
            <DigestMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <DigestValue>
                g1QqF2amKSwrGc4EHjgvhpvI4jE=
            </DigestValue>
          </Reference>
        </SignedInfo>
        <SignatureValue>
            KlJ1j0…POB6KzPO00=
        </SignatureValue>
        <KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference
                URI=
                   "#SecurityToken-1ecd5e58-9ba6-4239-a9fb-
250e15dc3769"
                ValueType=
                   "http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-x509-token-profile-1.0#X509v3" />
          </wsse:SecurityTokenReference>
        </KeyInfo>
      </Signature>
    </wsse:Security>
  </soap:Header>

<soap:Body
    wsu:Id="Id-42767f08-d88f-4394-a4f3-06bf14b00916">
    <submitOrder
        xmlns="http://wss.samples.microsoft.com">
        <OrderImpl_1 xmlns="">
          <accountNumber>LUCKY-200-300-101</creditCardExpM>
          <quantity>200</quantity>
          <supplierItemNumber>99882388</supplierItemNumber>
          <SKU>2233888</SKU>
        </OrderImpl_1>
      </submitOrder>
</soap:Body>

</soap:Envelope>
</log>
```

Benefits and Limitations

Secure Object Handler is a generic implementation approach to abstract common security processing logic and decouple from the application processing logic. It can be used with various Java EE .NET interoperability technologies and has the following benefits:

Maintainability

Common security processing logic can be refactored into Secure Object Handlers to perform pre-processing or post-processing tasks. Developers just need to maintain the handlers centrally instead of modifying massive numbers of applications that have security processing logic embedded.

Scalability

The Java EE .NET interoperability solution can be more manageable if the security processing logic is decoupled from the business processing logic. This allows Java and .NET applications to extend their security processing logic by either extending existing handler functionality or adding a chain of handlers (if the underlying platform or product supports the handlers).

Limitations

There are a few limitations when using handlers. First, there may be existing legacy applications that do not support the use of handlers. For example, .NET applications that do not use WSE do not benefit from the WS-Policy framework. Developers have to modify the applications to add custom handlers, which may cause considerable change on impact and implementation risks.

Second, some integration strategies' implementations can only support one single handler and not multiple handlers. Thus developers need to consolidate all security processing logic into a single handler. This will overload the handler design in one single implementation.

Fisher_CH13 3/24/06 5:05 PM Page 475

Secure Tracer Strategy

Problems

Audit control and compliance requirements always expect a good collection of transaction logging information. Sometimes a copy of the SOAP messages may be retained so that raw data can be available for complicated security analysis and reporting when needed. In practice, too much data may not be helpful if administrators need to find out whether there is any suspicious identity theft from signs of identity spoofing or man-in-the-middle attacks. Further, many security audit reports do not correlate the business transactions into sender/recipient pairs or provide the flexibility to apply specific security processing rules, such as a recipient's message timestamp should be later than the sender's message timestamp.

If the sender and the recipient are operating on different platforms, the security reporting and tracing of suspicious business transactions are more complicated because administrators need to collate the service requester and the response messages together to correlate them. If the business data are encrypted, administrators are not able to scan and trace raw data manually. They need to rely on some tools to identify whether there are any sender's or recipient's messages that have invalid user credentials, such as invalid X.509 certificates, or suspicious message timestamp, for example, recipient's message timestamp is earlier than sender's message timestamp, which may suggest a message alteration or spoofing.

The Secure Tracer strategy is intended to address the challenge of tracing business transactions for suspicious activities. It allows administrators to correlate the sender's and recipient's messages in pairs and applies some security processing rules to identify any suspicious messages or to trace the source details from the central logging repository.

Solution

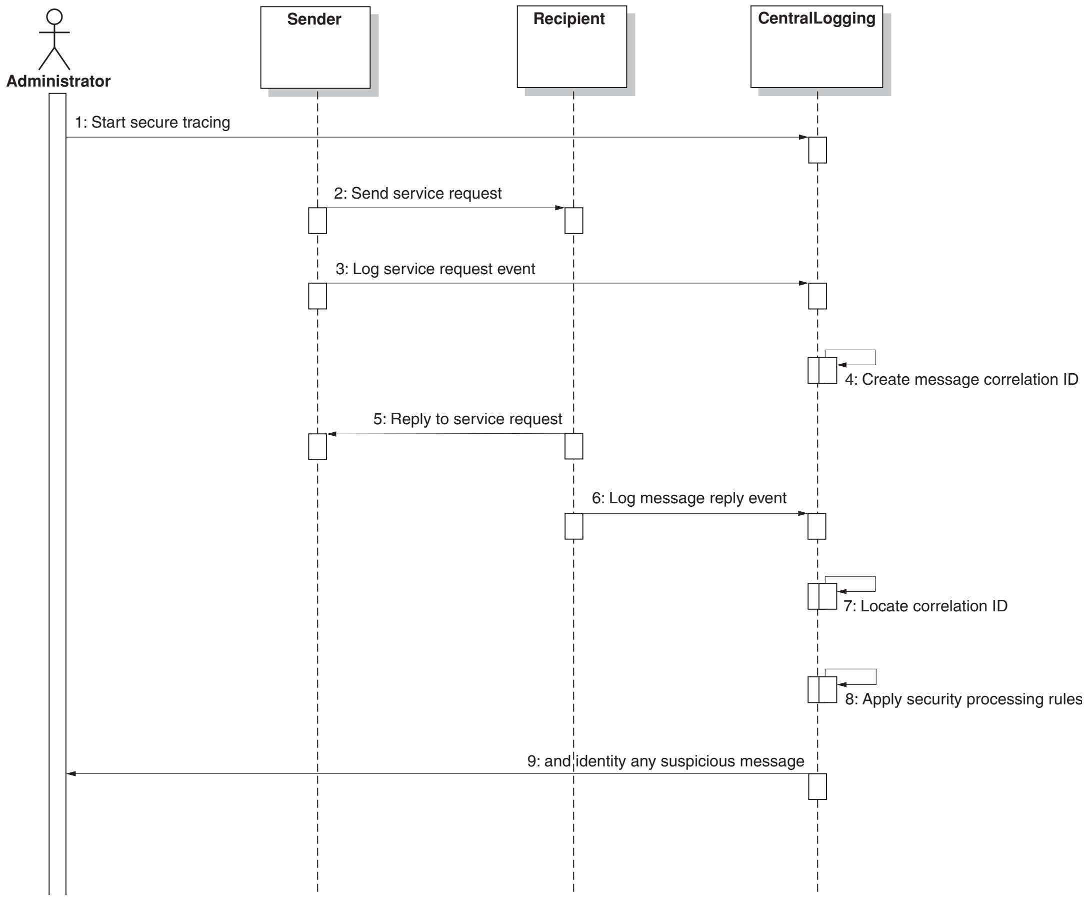
The Secure Tracer Strategy applies to both synchronous, asynchronous, and bridge interoperability strategies that utilize Web services. It intercepts service requests and replies in SOAP messages (whether synchronous or asynchronous) and correlates them in pairs. This also includes bridge strategy that supports Web services. In either strategy, a central logging mechanism needs to be in place.

The secure tracer strategy depends on a central logging mechanism to pull both the sender and the recipient log messages together. This can be implemented by a variety of middleware such as a basic file transfer facility, ftp, for example, or ESB.

Once the central logging mechanism is implemented, a simple message matching application can be built to correlate the sender and the recipient messages that refer to the same service request together. Figure 13-8 depicts a sequence diagram that illustrates a secure tracer strategy.

- Administrators start secure tracing activities (Step 1).
- Upon sending the service request (Step 2), the Sender issues a log message to the central logging system (Step 3).
- Central Logging creates a message correlation identifier for easy identification and mapping of the service request and the corresponding reply messages (Step 4).
- Upon completion of processing the service request (Step 5), the Recipient issues a log message to the central logging system (Step 6).
- The central logging system correlates the sender's message with the recipient's message by message id, message timestamp, and reference and creates a grouping between the two (Step 7).
- The central logging system also applies some security processing rules to ensure the message is authentic and genuine (Step 8). For example, it verifies the digital signature of both the sender's and recipient's messages and checks whether the recipient's timestamp is earlier than the sender's timestamp.
- Suspicious messages are flagged for administrator's attention by putting these messages under the alert queue (Step 9). Optionally, the central logging system should notify the administrator by e-mail or other predefined communication mechanism (such as fax or instant messaging).

Fisher_CH13 3/24/06 5:05 PM Page 477

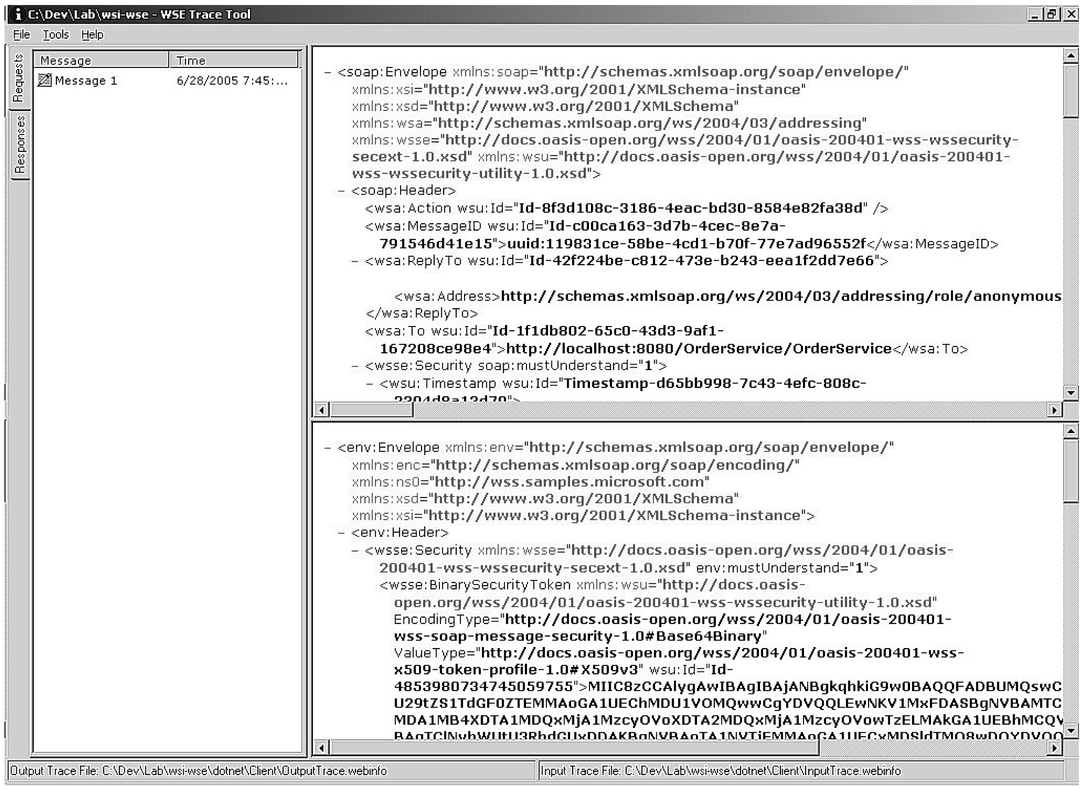


Administrator
Sender
Recipient
CentralLogging
1: Start secure tracing
2: Send service request
3: Log service request event
4: Create message correlation ID
5: Reply to service request
6: Log message reply event
7: Locate correlation ID
8: Apply security processing rules
9: and identity any suspicious message

Figure 13-8
Secure Tracer sequence diagram

Example

Figure 13-9 shows WSE Trace Tool (downloadable from www.gotdotnet.com/workspaces/workspace.aspx?id=ab938e2f-cabf-4145-b0e9-dbeeaf51dbe5), which correlates the incoming service request message with the outgoing reply message. It is not an implementation of the secure tracer strategy, but it illustrates a good example of correlating the service request and reply messages for manual inspection.



```
- <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
    wss-wssecurity-utility-1.0.xsd">
  - <soap:Header>
      <wsa:Action wsu:Id="Id-8f3d108c-3186-4eac-bd30-8584e82fa38d" />
      <wsa:MessageID wsu:Id="Id-c00ca163-3d7b-4cec-8e7a-
        791546d41e15">uuid:119831ce-58be-4cd1-b70f-77e7ad96552f</wsa:MessageID>
    - <wsa:ReplyTo wsu:Id="Id-42f224be-c812-473e-b243-eea1f2dd7e66">

          <wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
      </wsa:ReplyTo>
      <wsa:To wsu:Id="Id-1f1db802-65c0-43d3-9af1-
        167208ce98e4">http://localhost:8080/OrderService/OrderService</wsa:To>
    - <wsse:Security soap:mustUnderstand="1">
      - <wsu:Timestamp wsu:Id="Timestamp-d65bb998-7c43-4efc-808c-
          2204d8a12d70">
```

```
- <env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns0="http://wss.samples.microsoft.com"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <env:Header>
    - <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
        200401-wss-wssecurity-secext-1.0.xsd" env:mustUnderstand="1">
        <wsse:BinarySecurityToken xmlns:wsu="http://docs.oasis-
          open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
          wss-soap-message-security-1.0#Base64Binary"
          ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
          x509-token-profile-1.0#X509v3" wsu:Id="Id-
          4853980734745059755">MIIC8zCCAlygAwIBAgIBAjANBgkqhkiG9w0BAQQFADBUMQswC
          U29tZS1TdGF0ZTEMMAoGA1UEChMDU1VOMQwwCgYDVQQLEwNKV1MxFDASBgNVBAMTC
          MDA1MB4XDTA1MDQxMjA1MzcyOVoXDTA2MDQxMjA1MzcyOVowTzELMAkGA1UEBhMCQV
          BAgTClNvbWUtU3RhdGUxDDAKBgNVBAgTA1NVTiEMMAoGA1UECxMDSldTMQ8wDQYDVQQ
```

Output Trace File: C:\Dev\Lab\wsi-wse\dotnet\Client\OutputTrace.webinfo
Input Trace File: C:\Dev\Lab\wsi-wse\dotnet\Client\InputTrace.webinfo

Figure 13-9
WSE Trace tool correlates the service requester and the response messages together.

Benefits and Limitations

Secure tracer strategy is a simple audit log reporting function that is intended to address basic security auditing, such as track and trace the secure transactions, and compliance requirements. It has the following benefits:

Support Compliance Requirements

With the changing requirements of local compliance, it is crucial to provide a secure tracing capability to meet audit control and compliance needs.

Fisher_CH13 3/24/06 5:05 PM Page 479

Proactive Reporting of Suspicious Activities

The capability to identify suspicious messages that may have been tampered with or to scan for any problematic digital signature in the messages can allow suspicious or potential security vulnerabilities to be detected proactively.

Limitations

The extensibility of the secure tracer implementation depends on the central logging mechanism and the processing logic for the message correlation. Not all messages can be correlated. For example, the secure tracer implementation has no clue how to correlate a collection of encrypted SOAP messages, given that the message identifiers and message digest do not show any correspondence.

Related Patterns

- **Chain of Responsibility Pattern** The Chain of Responsibility pattern (refer to [CoR1] and [CoR2] for details) allows a handler object to handle the service request without coupling the sender of the service request to the recipient. By using the handler object, this pattern chains together the receiving objects and passes the service request along the chain of processes. The Chain of Responsibility pattern is generic to Java or .NET platforms and can be applied to security processing logic specific to Java EE .NET interoperability, assuming that the technical details need to be hashed out.
- **Message Inspector Pattern** The Message Inspector pattern (refer to [CSP] for details) introduces the concept of a message handler chain of security processing actions to pre-process and post-process SOAP messages. These actions can include verifying user identity, validating messages for compliance with Web services standards, validating digital signatures, encrypting and decrypting business data, and auditing and logging. This design strategy is specialized for Web services running on the Java platform and does not cover the details of Java EE .NET interoperability or related technologies.

- **Secure Logger Pattern** The Secure Logger pattern (refer to [CSP] for details) introduces some best practices to create secure logs for business transactions using message digest, cipher, signature, and UID generator classes. This pattern is targeted for Java applications developed for the Web tier and does not cover the details of Java EE .NET interoperability or related technologies.

Best Practices and Pitfalls

The following recapitulates some best practices and pitfalls regarding the use of WS-Security for interoperability.

Best Practices

- Use compatible or certified software component versions. Don't assume the latest version of open source components always work with the existing code base.
- Use specific encryption and digest algorithms that are proven to work for Java EE .NET interoperable products. Use Triple DES for session key encryption to enable WSE in the `app.config` for interoperability.

```
<binarySecurityTokenManager
   valueType=
   "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-
profile-1.0#X509v3">
   <sessionKeyAlgorithm name="TripleDES" />
</binarySecurityTokenManager>
```

- Use Optimal Asymmetric Encryption Padding (RSAOAEP RSA) as the encryption key algorithm to enable WSE in the `app.config` file for interoperability.

```
<binarySecurityTokenManager
   valueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3">
   <keyAlgorithm name="RSAOAEP" />
</binarySecurityTokenManager>
```

Fisher_CH13 3/24/06 5:05 PM Page 481

- Use built-in handlers or security policies wherever possible, instead of rewriting your own security processing logic. If you want to customize your own security processing logic, you may consider extending the existing handlers.

Pitfalls

- **Certificate management on both platforms can be problematic** For example, if the digital certificate is expired, the error messages may disguise the problem as being invalid credentials or keys but not the expired certificates.
- **Use of security exception** Don't just catch the exception. Make the error message meaningful. For example, the exception "no policy found" can be ambiguous and does not tell what the root cause is.

Summary

Interoperability for security is a challenging subject, and dealing with security on one single platform is already complex. Making two different application platforms interoperate on the security is more complicated. Thus adopting open standards for security interoperability would be a good approach. Web services security certainly is a turning point for Java EE .NET security interoperability. With the availability of Web SSO MEX specification, security interoperability would be viable for users to keep the best of both identity federation infrastructures.

There are many reference materials and documentation about Java EE and .NET security. Nevertheless, the availability of references for the security interoperability is limited probably because the interoperability standards and the supporting technologies are evolving. The good news is that free interoperability software kits (such as WSE and JWSDP) are available for public download.

In this chapter, Secure Object Handler and secure tracer strategies introduce essential best practices to managing interoperability. These two strategies can be implemented in both Java and .NET platforms.

Fisher_CH13 3/24/06 5:05 PM Page 482

References

[Anne2] Anne Anderson. "IEEE Policy 2004 Workshop 8 June 2004—Comparing WSPL and WS-Policy." IEEE Policy 2004.

http://www.policy-workshop.org/2004/slides/Anderson-WSPL_vs_WS-Policy_v2.pdf

[Anne3] Anne Anderson. "An Introduction to the Web Services Policy Language (WSPL)." Sun Microsystems Laboratories, 2004.

http://research.sun.com/projects/xacml/Policy2004.pdf

[BSP] Web Services Interoperability Organization. Basic Security Profile Version 1.0. Working Group Draft. May 15, 2005.

www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

[CoR1] Mark Grand. "Pattern Summaries: Chain of Responsibility."

www.developer.com

and

www.developer.com/java/other/article.php/631261

[CoR2] data & object factory. "Chain of Responsibility."

www.dofactory.com/Patterns/PatternChain.aspx

[CSI] Computer Security Institute. CSI/FBI Computer Crime and Security Survey. Computer Security Institute 2005.

[CSP] Chris Steel, Ramesh Nagappan, Ray Lai. *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Boston: Prentice Hall, 2006.

[J2EE14] Bill Shannon. "Java^{TM} 2 Platform Enterprise Edition Specification, v1.4." Proposed Final Draft 3. Sun Microsystems, April 2003.

http://java.sun.com/j2ee/j2ee-1_4-pfd3-spec.pdf

[J2EE14Tutor] Eric Armstrong, et al. "The J2EE 1.4 Tutorial." Sun Microsystems, 2003.

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/

[JavaVMFlaw] Security Focus. "Security Vulnerability in Sun's Java Virtual Machine Implementation." Security Focus. October 23, 2003.

www.securityfocus.com/archive/1/342147

[LibertyWSFed] Liberty Alliance. "Liberty Alliance & WS-Federation: A Comparative Overview." Liberty Alliance Project White Paper. October 14, 2003.

www.projectliberty.org/resources/whitepapers/wsfed-liberty-overview-10-13-03.pdf

[LiGong] Li Gong. "Java Security Architecture." in "Java 2 SDK, Standard Edition Documentation Version 1.4.2." Sun Microsystems, 2003.

http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/securityspec.doc1.html

and

http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/securityspec.doc2.html.

[SGuest] Simon Guest. "WS-Security Interoperability Using WSE 2.0 and Sun JWSDP 1.5." Microsoft, May 2005.

http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnbda/html/wssinteropjwsdp15.asp

[Watkins] Dr. Demien Watkins. "An Overview of Security in the .NET Framework." MSDN Library, January 2002.

http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnnetsec/html/netframesecover.asp

[WindowsAuthFlaw] Microsoft. "MS02-011: An Authentication Flaw Could Allow Unauthorized Users to be Authenticated on the SMTP Service." Article 310669. Revision 7. Microsoft Support, April 13, 2004.

http://support.microsoft.com/default.aspx?scid=kb;EN-US;Q310669

[WSI-countermeasure] Jerry Schwarz, et al., ed. "Security Challenges, Threats and Countermeasures Version 1.0" Final Material. Web Services Interoperability Organization, May 7, 2005.

http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.0.pdf

[XACML2] OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0. February 1, 2005.

http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-ALL.zip

Fisher_CH13 3/24/06 5:05 PM Page 484